1995-06

A client/server application development methodology for DoD

McDermitt, David R.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

A CLIENT/SERVER APPLICATION DEVELOPMENT METHODOLOGY FOR DOD

by

David R. McDermitt

June, 1995

Thesis Advisors: James C. Emery
Magdi N. Kamel

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE June, 1995	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE A CLIENT/SERVER APPLICATION DEVELOPMENT METHODODLOGY FOR DOD	5. FUNDING NUMBERS
6. AUTHOR McDermitt, David R.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

Since the Department of Defense began utilizing electronic computers in the 1950's; it has been plagued by inefficient software development practices. The result of such practices was software that was over budget, behind schedule, of poor quality, and one that usually did not satisfy user requirements. The rapid pace with which business needs are changing, particularly within DoD, demand that software be developed under even tighter schedules than have been experienced before.

In order to respond to the demand and provide a product which meets budgetary and schedule constraints without sacrificing quality, it is absolutely necessary that DoD adopt modern methods and practices for developing software.

This thesis presents a modern methodology for developing applications in a client/server environment. The methodology is based on a combination of modeling and prototyping to deliver quality applications quickly and inexpensively. An example application was developed based on the proposed methodology and serves as a model for migrating legacy DoD mainframe applications to modern client/server technology. It is hoped that this process will serve as an example of how DoD can benefit from modern development strategies and tools.

14. SUBJECT TERMS Client/Server, Application Development, Legacy Systems, System Migration.			15. NUMBER OF PAGES 86
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)

Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited.

A CLIENT/SERVER APPLICATION DEVELOPMENT METHODOLOGY FOR DOD

David R. McDermitt
Lieutenant, United States Navy
B.S., Auburn University, 1989

Submitted in partial fulfillment
of the requirements for the degree of

**MASTER OF SCIENCE IN
INFORMATION TECHNOLOGY MANAGEMENT**

from the

**NAVAL POSTGRADUATE SCHOOL
June, 1995**

Author: [redacted]
David R. McDermitt

Approved by: [redacted]
James C. Emery, Thesis Co-Advisor

[redacted]
Magdi N. Kamel, Thesis Co-Advisor

[redacted]
David R. Whipple, Chairman
Department of Systems Management

ABSTRACT

Since the Department of Defense began utilizing electronic computers in the 1950's, it has been plagued by inefficient software development practices. The result of such practices was software that was over budget, behind schedule, of poor quality, and one that usually did not satisfy user requirements. The rapid pace with which business needs are changing, particularly within DoD, demand that software be developed under even tighter schedules than have been experienced before.

In order to respond to the demand and provide a product which meets budgetary and schedule constraints without sacrificing quality, it is absolutely necessary that DoD adopt modern methods and practices for developing software.

This thesis presents a modern methodology for developing applications in a client/server environment. The methodology is based on a combination of modeling and prototyping to deliver quality applications quickly and inexpensively. An example application was developed based on the proposed methodology and serves as a model for migrating legacy DoD mainframe applications to modern client/server technology. It is hoped that this process will serve as an example of how DoD can benefit from modern development strategies and tools.

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

During this decade American society has experienced a transformation unlike any seen before. Organizations of all kinds are forced to do more with fewer assets if they expect to remain competitive in the market place. Nowhere is this more true than in the Department of Defense.

It is generally agreed that, in order to survive, organizations have to re-think the way they do business. Organizational downsizing and the distribution of responsibilities are common themes in response to the tremendous global changes that surround us. As budgets and spending continue to shrink, it is increasingly necessary to push decision making power out to the organization's periphery. This empowerment requires that people at all levels of the organization be provided with appropriate tools to help them in their daily decision making processes.

In many ways, the evolution of information systems architectures mimic the evolution of the organizations that they serve. Since the 1950's computers have become increasingly important for the conduct of business, in industry and in defense.

In the 50's and 60's when most organizations were highly centralized, the mainframe was the dominate fixture on the corporate computing landscape. In the 70's and 80's, as organizations spread out to include national and international divisions, it became increasingly difficult to request reports and data services from a centralized MIS group. The mainframe continued to thrive as did the central bureaucracy of most large organizations; but because information in corporate databases was difficult to access and use, end-users supplemented mainframes with desktop computing using personal computers.

In response to a growing demand for speedy, ad-hoc reports and easy to use computer applications, self-contained "stovepipe" information systems were developed to enable corporate divisions to get their work done. The proliferation of powerful desktop computers and easy to use database and spreadsheet software aided in driving the PC to a position of prominence in corporate computing, a position it still occupies today.

As organizations have decentralized in the 1990's, so have their information systems. The power available on desktop computers makes attractive presentation of information possible, and the availability of computing power makes it logical to share the burden for processing the information.

This thesis is about providing end-users with powerful computing solutions that access worldwide data and provide answers to complex business problems. It examines how applications are developed that take advantage of distributed processing, computers connected via networks, and shared information in corporate databases. The thesis also addresses the issues of maintaining control over dispersed applications and their users.

B. OBJECTIVES

The primary objective of this research is to lay the foundation for thinking about client/server system development within the Department of Defense. Provisionally, the case study, including the subsequent system design and implementation, provide the Human Resources Office of the Naval Postgraduate School with a maintainable, state-of-the-art client/server system. This thesis argues that the HRO system serves as a model for other commands in migrating from existing, monolithic applications to contemporary, distributed architectures.

C. SCOPE, LIMITATIONS, AND ASSUMPTIONS

The scope of this project is limited to new application development. In so far as existing applications are able to be modeled as new development, they will be discussed.

The HRO case study is an example of this type. Because the actual implementation of the solution involves so much new development, the project can be treated as a new system.

It is assumed in this thesis that the reader have a strong grasp of computer systems — in particular, a general working knowledge of client/server systems, database systems, and computer programming is assumed.

D. DEFINITIONS AND ABBREVIATIONS

The following are acronyms used in this thesis

BPR	Business Process Re-engineering
C/S	Client/Server
COBOL	Common Business Oriented Language
CRT	Cathode Ray Tube
DCPDS	Defense Civilian Personnel Data System
DIN	Data Identification Number
DoD	Department of Defense
ESP	Electronic System for Personnel
HRO	Human Resources Office
IS	Information Systems
IT	Information Technology
Kbps	Kilobits per Second
LAN	Local Area Network
Mbps	Megabits per Second
MIS	Management Information Systems
MS-DOS	Microsoft Disk Operating System
NOS	Network Operating System
NPS	Naval Postgraduate School
OLTP	On-Line Transaction Processing
OPM	Office of Personnel Management

PC	Personal Computer
RAD	Rapid Application Development
SQL	Structured Query Language
UI	User Interface
WAN	Wide Area Network

E. ORGANIZATION

Chapter II provides a brief discussion of the technology that serves as a backbone for client/server applications. The chapter attempts to provide a definition for client/server computing and includes sections on the general benefits and costs of implementing client/server solutions. The chapter further discusses the different architectures on which a client/server system might be based. It closes with factors to be considered when developing applications for deployment in a client/server environment.

Chapter III presents a systematic methodology for the analysis, design, and implementation of client/server applications. The chapter focuses on the analysis and design of the data and the functionality to be provided. Unique to the client/server environment are decisions regarding the allocation of data and functionality and where they should be located. These considerations are discussed in depth in the design of functional and data "partitions."

Chapter IV is a case study that tracks the development of a real-world client/server application, developed by the author for use at the Naval Postgraduate School. The case study is particularly relevant in its treatment of the migration of a system from a monolithic mainframe application to a contemporary client/server architecture.

Chapter V presents conclusions and makes recommendations for further research in this area.

The appendix provides images of the user interface of the case study. Complete source code listings are provided as additional documentation.

II. OVERVIEW OF CLIENT/SERVER COMPUTING

A. WHAT IS CLIENT/SERVER COMPUTING?

In the past, information systems that allowed multiple users to access central data were designed for use on mainframe computers. The explosion of computing power available on the desktop has provided a more efficient means for accessing central data. Modern information systems that allow multiple users to access central data consist of applications that call for data in addition to the actual data repository. In addition to the physical separation of the application and the data, most modern approaches separate processing as well. Normally there is a *physical* separation, but it is the *logical* separation of processing and data that forms the key to client/server. In a client/server environment, the storage of data and the processing power are shared among *clients* and *servers*. Simply stated, clients are processes that request service from other processes. Servers are the processes that provide that service. It is possible that servers may also be clients, but most clients are generally not servers.

The Gartner Group, an information technology research consultancy, have developed a popular model for defining the layers that make up various versions of client/server information systems. These layers segregate the activities of the system and define clearly the processing activity. According to this model, the layers are the presentation layer, the application logic layer, and the data management layer.

The presentation layer is solely responsible for generating the user interface for the application. It receives information from the application logic layer and displays it to the user in a format defined by the application. User input is also handled by the presentation layer. In client/server systems all presentation layer processing is usually performed by the client.

The application logic layer contains the "intelligence" of the program. Application logic is responsible for interpreting user input prior to making requests of the server and for manipulating data after it has been received from the server. Application logic is frequently shared between clients and servers to optimize performance of the application.

The data management layer is focused on efficiently responding to requests for data. This is a server function, regardless of the physical location of the database management system.

The Gartner Group further observes that client/server systems can be classified according to the allocation of tasks among the clients and servers. Its classification scheme, known as the "five strategies model," details the distribution of each of the layers. (see figure 1)

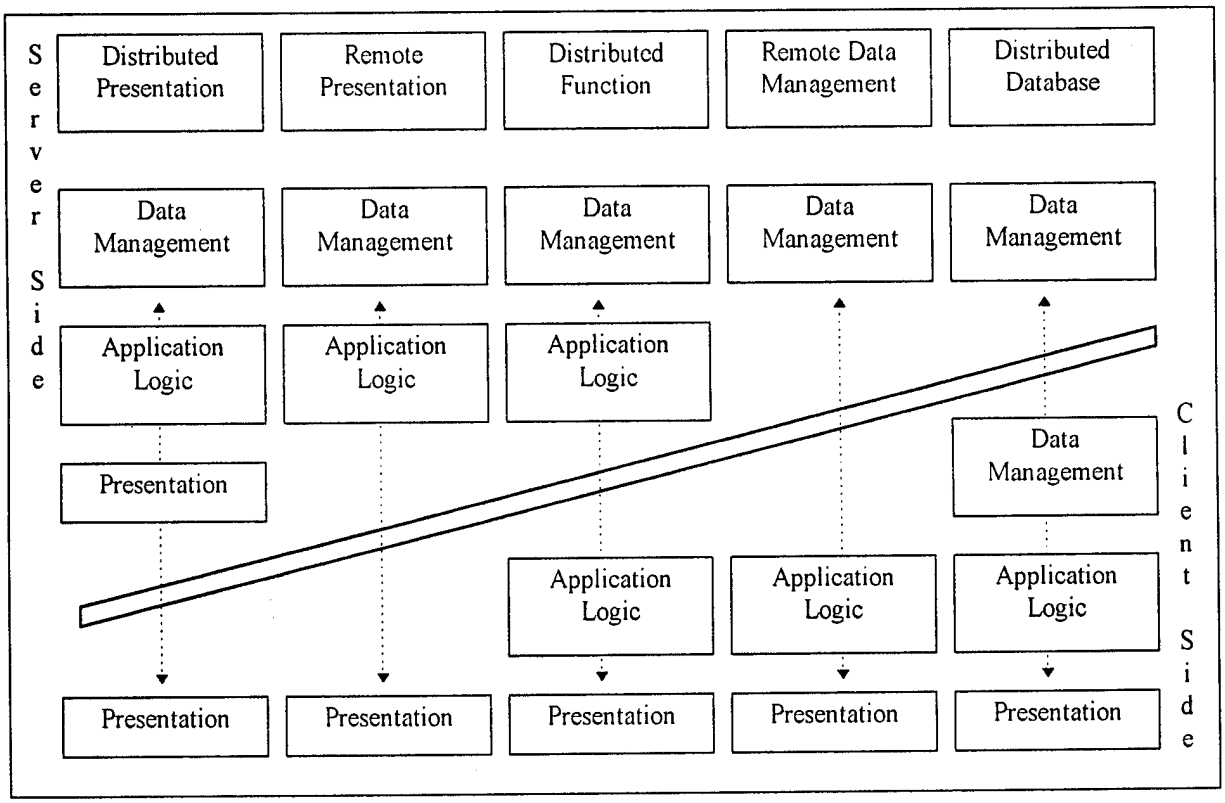


Figure 1 : The Five Strategies Model
Partitioning functionality between client and server platforms.
[Ref. 1]

Robert Orfali further defines client/server systems as those possessing the following characteristics [Ref. 2]:

- *Service*: Client/Server is primarily a relationship between processes running on separate machines.
- *Shared resources*: A server can service many clients at the same time and regulate their access to shared resources.
- *Asymmetrical protocols*: There is a many-to-one relationship between clients and server.
- *Transparency of location*: Client/Server software usually masks the location of the server from the clients.
- *Mix and match*: Software is independent of hardware or operating system software platforms.
- *Message-based exchanges*: Loosely coupled systems which interact through a message-passing mechanism.
- *Encapsulation of services*: A message tells a server what service is requested; it is then up to the server to determine how to get the job done.
- *Scaleability*: Client/Server systems can be scaled horizontally or vertically. Horizontal scaling means adding or removing client workstations with only a slight performance impact. Vertical scaling means migrating to a larger and faster server machine or multiservers.
- *Integrity*: The server code and server data is centrally maintained, which results in cheaper maintenance and guarding of shared data integrity. At the same time, the clients remain personal and independent.

B. COSTS AND BENEFITS OF CLIENT/SERVER COMPUTING

Little concrete data exists to document the costs of implementing a client/server architecture. As more organizations implement client/server solutions, they discover that these solutions may not represent a cost savings as originally anticipated. What many organizations *are* discovering is that the additional costs of doing business in the client/server world pay off in intangible ways.

Research recently conducted by the Gartner Group reveals that the costs involved in implementing client/server solutions may be two or three times that of a comparable

mainframe solution. Due to the fact that almost all mainframe costs can be isolated in the information systems department, it is easier to quantify those costs. In client/server implementations, costs are fragmented throughout the organization, often not identified explicitly as IT costs. As a result, costs are generally very difficult to quantify.

Additional costs include increased training, development demands, system and application maintenance, and system planning. It is estimated that over 70 percent of the costs involved in a client/server system are labor costs, including end-user labor (41%), end-user support labor (15%), application development labor (8%), and server operation labor(8%). This compares with the meager expenditures for client and server hardware, which represent only 18% of the total expenditure.[Ref. 3]

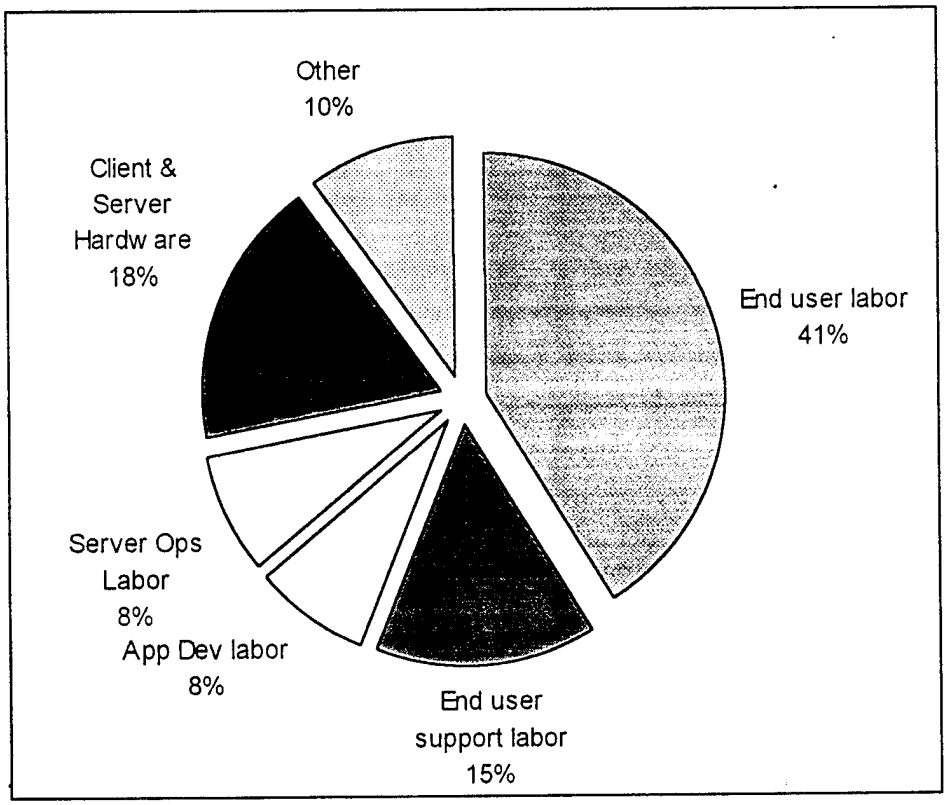


Figure 2 : Client/Server Expenses
[Ref. 3]

The benefits that most organizations experience are in the form of more effective information sharing, easier modifications to changing business environments, and improved customer service. While the costs of implementing client/server may be high, the costs of ignoring the benefits are certainly higher. The market, at least, judges the client/server architecture to be attractive. For example, Dataquest, the San Jose based market research firm, reports that firms in the United Kingdom with full or partial client/server information systems will experience a growth rate of 80 percent or more between now and the year 2000.

C. CLIENT/SERVER ARCHITECTURES

As depicted in Table 1, the components of client/server have migrated between client and server while still being considered part of the formal client/server model. Prior to the release of the Gartner Group model, the lack of a defined client/server implementations led to considerable confusion. Vendor claims have sustained and added fuel to the argument. For purposes of this discussion, the type of service provided by the individual software solution will dictate what type of client/server architecture is implemented.

1. File Server

In a file server implementation, the server receives requests from clients for individual files. The server, in turn, returns to the client the requested file for processing by the client. The server takes no active role in the processing of the data file; it acts only as a repository for information stored as separate files.

2. Database Server

A database server provides the client with the ability to request data using SQL statements that are processed by the server. The server returns to the client selected results from each SQL statement received from the client. The client runs an application that generates the SQL statements based on inputs from the user and then presents the

returned data back to the user in a useful format (which may require considerable processing power on the client to transform server data into useful information). Database servers are most commonly used in decision support systems where most queries are generated by the user on a client machine at run time.

3. Transaction Server

A transaction server is similar to a database server except that the SQL statements are stored on the server and compiled as a stored procedure or transaction. The transactions are *triggered* by the client application and then data is supplied to the client for further processing or display. Stored procedures offer tremendous performance advantages due to their compiled nature, and should be used whenever practicable to boost performance of the overall system. These types of applications are generally referred to as On-Line Transaction Processing (OLTP) systems and are typically used in implementations where response time is critical.

4. Groupware

Groupware is unlike other forms of client/server systems. Popularized by the commercially successful Lotus Notes™ product, groupware is a way for groups of individuals to share information and automate the flow of work. Groupware is unique in the types of data and information that it allows to be shared. Utilizing products like Lotus Notes™, it is possible for physically dispersed individuals to collaborate on common work projects. Shared access to complex data and information is made easier through the use of these proprietary systems.

5. Distributed Objects

The most recent innovation in client/server architectures involves independent objects that communicate to accomplish a task. These "distributed objects" encapsulate the complexity of their processes and communicate with each other via a set of standardized methods. Client objects and server objects exchange requests and responses

through the use of an object request broker (ORB). The ORB handles the details of locating the required objects and timing of message dispatch.

D. PARTITIONING THE CLIENT/SERVER APPLICATION

In any client/server application it is necessary to decide which platform will "own" certain processes. Will it be the client or the server? In a multi-server environment, whose server will own the process?

1. Server-Based Processing

One of the main advantages of having a client/server system is that the work load can be shared. The decisions that determine how the processing will be shared generally occur through a process called "tuning". Tuning identifies performance-critical portions of the application and locates those procedures on either the client or the server to maximize performance. By centrally locating business logic on the server, modifications to that logic can be implemented quickly, easily, and transparently. However, the availability of powerful and inexpensive processors in many clients force careful consideration of the trade-offs involved.

The most common implementation of server-based business logic is stored procedures. Because they are typically compiled, stored procedures allow for fast execution and higher relative transaction rates.

It is important to understand that stored procedures are not a panacea. If it is necessary to implement logic that cannot be expressed in database calls or the data resides on heterogeneous servers, then stored procedures may be a partial solution that must be supplemented by additional processing on the client.

2. Client-Based Processing

The most common use of client-based processing is in the implementation of the graphical user interface, or GUI. Because the GUI represents the "look and feel" of the entire application, a successful GUI design will frequently determine the overall success of

the application. The GUI will generate requests for the server based on input received from the user. In addition to GUI processing, local business rules must also be processed by the client.

3. Cooperative Processing

Depending on the implementation selected, some processes must be shared across the network between the client and the server. Special attention should be paid to the design of the shared processes to limit communication between the client and server portions. The overhead involved in extensive inter-process communications could easily degrade performance to an unacceptable level, as well as adding significantly to communication costs.

E. COMPONENTS OF CLIENT/SERVER APPLICATIONS

The overall client/server application has three main players: the client, the server, and the middleware. These three separate components must be chosen carefully to ensure that they will be interoperable and that they will support the collaboration necessary to achieve the desired result. This section will discuss each component of the client/server environment in some detail.

The discussion in this section relies heavily on material from Robert Orfali's *Essential Client/Server Survival Guide*, an excellent introduction to the concepts necessary to understand client/server systems. [Ref. 2]

1. The Client

a) Operating System

The type of operating system (OS) running on the client will determine what type of user interface the application will support. All operating systems must provide methods to allow requests and replies to be processed. In addition, some type of file transfer mechanism is necessary. Services that are highly desirable include preemptive multitasking,

priority tasking, intercommunication processes, and multiple threads. These features assist in the efficient utilization of the client in handling the replies from the server.

b) Interface Concerns

Depending on the application, a GUI may or may not be required. In simple applications where clients are being used for nothing more than data entry and perhaps limited user feedback, a full-blown GUI may be overkill. Furthermore, a client that relies on a GUI may leave the user in an undesirable position if it is not properly implemented, following strict design guidelines. Consistency in UI design is frequently overlooked.

c) Application Design

The design of the application is based on the business needs that the application is intended to fulfill, the data that the application will have available, and the information that is expected from the application. Organizations must adapt quickly to changing market conditions to remain competitive. The rapidly shifting requirements that businesses experience dictate short development cycles.

The process that has proved most advantageous is the Rapid Application Development cycle. This development cycle focuses on quickly developing working prototypes that deliver basic functionality, reviewing the design often, discussing potential improvements, and delivering the application. The greatest benefit of RAD is its extreme flexibility, hence allowing developers to deliver higher quality software in shorter periods of time.

d) Development Tools

The development tools that are selected depend to a great extent on the client operating system. For example, under Microsoft Windows there are many development environments that make designing, developing, and delivering software efficient and cost-effective. Fewer tools are available for other operating systems because the market force has tended to focus attention on the Microsoft Windows platform.

2. The Middleware

a) Transport Stacks

Transport stacks are the communications standards that govern the way computers can talk to each other. The choice of transport stack depends on the type of operating system, the Network Operating System and whether a Wide Area Network (WAN) or Local Area Network (LAN) will be used. Some of these choices include TCP/IP, NetBIOS, IPX/SPX, DECnet, OSI, and AppleTalk.

b) Network Operating Systems

There are three main functions of Network Operating Systems(NOS). The NOS extends the local operating system's functionality by allowing access to networked devices such as printers, file directories, and modem pools. Additionally, the NOS makes the connection to these remote devices appear *transparent*, meaning that they appear to the user to be just like devices that are connected directly to the user's computer. The third function provided by the NOS is to allow for coordination of applications that are split across client/server lines.

c) Service-Specific Middleware

Service-specific middleware is the layer that is responsible for managing tasks that are specific to a service offered by the server. The most widely recognized service-specific middleware standards include:

- Database middleware
- Transactional Remote Procedure Call middleware
- Groupware middleware
- Object middleware
- Distributed System management middleware

3. The Server

The software that residing on the server system enables it to perform functions necessary to provide services requested from it. The remaining paragraphs in this section describe the components of software that typically reside on the server.

a) Operating System

All operating systems are made up of two types of service, base and extended. Base services are those which are a part of the core, or kernel, of the operating system. Extended services provide additional functionality in the form of "add-ins." Extended services or extensions are typically modular pieces of software that can be "snapped" together to provide the exact mix of services required for a particular server. [Ref. 2]

b) Network Operating System

The network operating system, or NOS, provides the developer with a programming interface that allows him to communicate with the network without having to address the hardware directly. The NOS, acting as an intermediary, eases the burden on the software developer because applications can be designed for the NOS application programming interface, or API. The API allows the application developer to utilize functionality built into the NOS without having to write computer code that performs every physical task that must be accomplished. This promotes hardware independence by providing a NOS and its associated API that will operate on many different hardware platforms.

c) Network Computing Environment

Hardware independence can be further promoted through support for standards that support the network computing environment. Several competing standards vie for market share in this competition to determine how applications that reside on physically distant machines of different hardware configurations will communicate. The Open Standards Foundation, or OSF, promotes the Distributed Computing Environment. Sun

Microsystems is promoting the Open Network Computing (ONC) standard. These two standards offer similar services, they differ in that DCE relies on CCITT protocol standards that are recognized worldwide while ONC relies on TCP/IP protocol standards, recognized most widely in the United States.

d) Network Management Environment

Because client/server systems are normally geographically dispersed, it is necessary to provide a means for managing remote systems. Utilizing network management software, system administrators can configure servers that are located anywhere. The three main products that are competing to become the standard are the Distribute Management Environment (DME), the Object Management Architecture (OMA), and the UI-Atlas. Each of these products has particular strengths and weaknesses, but generally DME is utilized by those choosing to adopt the OSI model, UI-Atlas by those selected Unix and TCP/IP, and OMA by those focusing on object request brokers.

III. CLIENT/SERVER DEVELOPMENT METHODOLOGY

A major benefit of client/server technology is that it enables a shift toward thinner layers of middle management and assist in empowering the workforce. The core of this technology changes how work is accomplished and who is considered a "user." In addition, the rapid growth in the volume of software demanded necessitates higher programmer productivity. These modifications require that development strategies shift to accommodate the new demand. Traditional software development techniques fail to provide an adequate methodology for guiding the development of these new types of architectures.

In addition to the shift in organizational climate that fosters the development of client/server systems, mission requirements in this new environment change rapidly with ever-shrinking project cycles. The development time for software using conventional methods often exceeds the cycle time of the changes. Adaptive and productive methodologies are required to permit business systems to keep up with business needs. Part of this process involves quickly producing working software that satisfies subsets of the overall requirements. These working models or prototypes are then enhanced and improved in an iterative fashion to produce robust applications that satisfy broader user needs. This process essentially continues "forever" as the organization continues to adapt to changes in missions, perceived user needs, and technology. The process is known as Rapid Application Development, or RAD.

The scale of the system to be developed will dictate whether using a formal or explicit methodology is necessary. There is considerable evidence that for small and medium-sized projects, formal methodologies stand in the way of building systems quickly and inexpensively, while best meeting the user's requirements. When does a system cease to be "medium sized?" This is a complex question that has no real answer, but certainly when multiple servers are involved, complexity will increase to a state where a

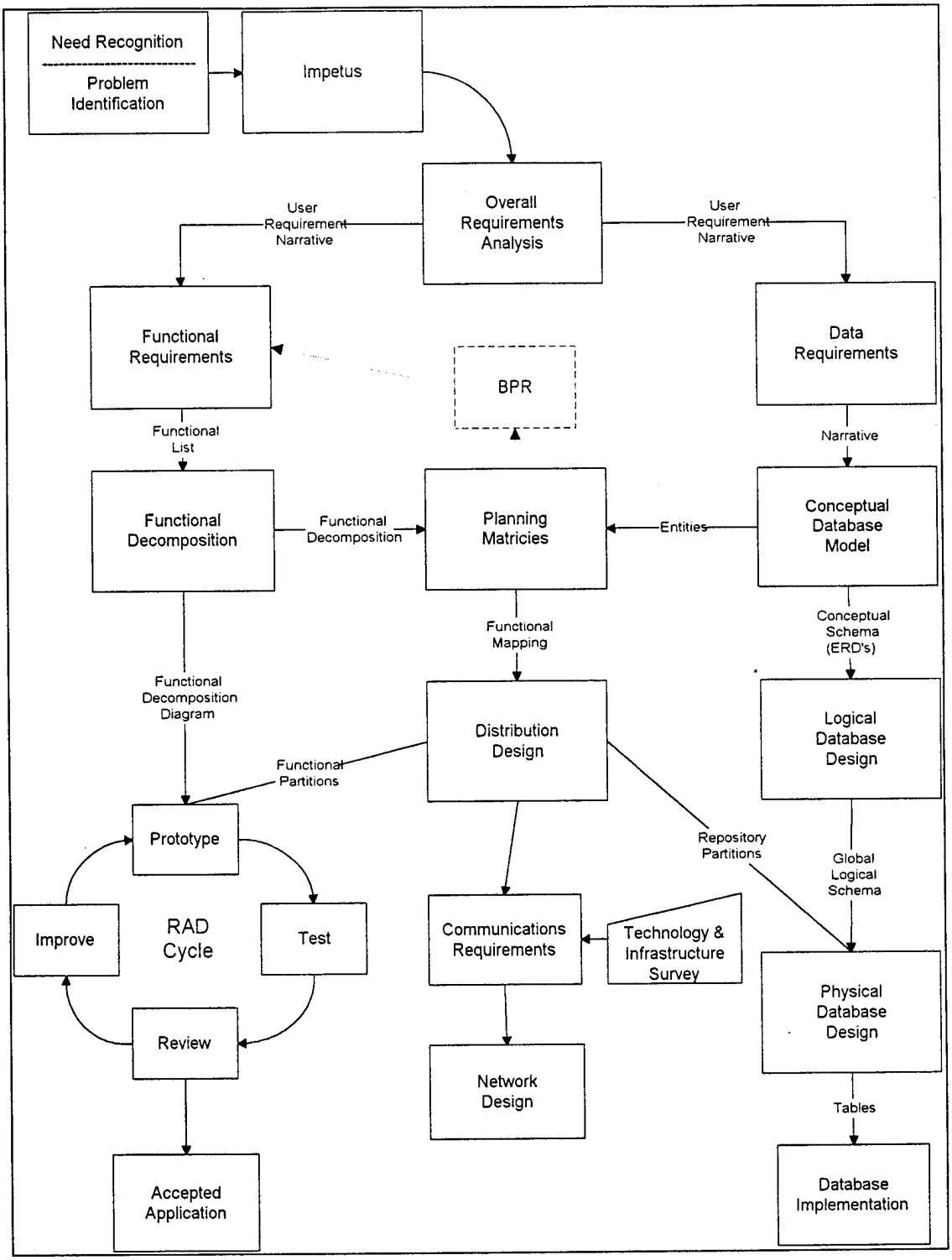


Figure 3 : Graphical Representation of Client/Server Development Methodology

formalized methodology becomes desirable.[Ref. 4]

The following sections present the phases and steps of a generic methodology. An overview of this methodology is presented as Figure 3.

A. REQUIREMENT ANALYSIS

Determining the requirements of the system is the first step in the development process. In a broad sense the requirements analysis provides a "birds eye" view of the system. It answers several general questions: What is to be provided? What must exist in order to meet the requirement? What processes are needed to transform available data into required information? In short, what are the inputs, outputs, and transformations that take place in the system.

In a client/server environment it is necessary to examine the system at three unique levels. First, the entire system should be analyzed to determine the overall requirements. The functional processes should then be examined to identify the "core" functionality that the solution must provide in order to be viable. Next, an analysis of the data should be undertaken to determine whether the proper data already exists or will have to be appended to meet the requirements of the functional analysis. Last, a survey of the organization's existing communications infrastructure should be conducted to highlight natural locations for data repositories.

1. Overall Requirements Analysis

The overall requirements analysis provides the application developer with his first exposure to the system. The goal of this analysis is to gain an overall understanding of the "application space," in other words, how will the application fit with the organizations current way of doing business. A main activity of overall requirements analysis is conducting interviews with management and end users.

a) Management Interviews

Prior to the start of the development process, management must recognize that a need exists. This need may reflect a deficiency in a current system or it may involve the

realization that a system of some type is required. Recognition of need and impetus to react on that need is assumed prior to the commencement of the development process.

Several factors must be identified at the earliest stage possible. These factors are: a) the overall plan for the application, b) how/if it will fit into existing systems, and c) what will determine the application's success. The importance of identifying these factors early is instrumental to the success of the project. This process will limit the amount of disagreement later in the development process.

The need for strong support from management cannot be overemphasized. Initial meetings with management are crucial to the success of the project. If management is excited at the prospect of improving or reinventing a system, that excitement will undoubtedly spread throughout the organization. This improves the chances for success of the overall system. Lack of management support is a prescription for certain failure.

b) End User Interviews

In a traditional system, end-user interviews are perhaps the most important stage in the development process. While these interviews are no less important in a modern methodology, a major difference between traditional and modern approaches is the iterative nature in which they occur. Traditional methodologies emphasize the need for meeting end-user requirements, but fail to account for the ever-changing nature of these needs. An application developed without user inputs throughout the entire development process is unlikely to represent the users requirements when it is delivered. An application developed in accordance with a rapid application development cycle continually revisits the suitability of the application to the users tasks.

During end-user interviews, it is vital to obtain as much information as possible about how the user currently performs a task. This may mean examining an existing system and obtaining forms, printouts, reports, manuals, regulations, etc... that describe how work is accomplished. The purpose of analyzing the existing system is not to duplicate the system, but to understand the fundamental needs of a process. Any analysis that spends more effort than required to do this will suffer from the proverbial "paralysis

from analysis."

End users are the individuals who will put the application to use daily. By reducing to writing their view of what the system should require and what it should provide, the application development team can compare this with the results of management interviews to determine the correct "mix" for the system. This mix will determine if the application serves as a seamless extension of work, making the task easier, or as an unwelcome requirement that serves as an irritation and a hindrance in performing the task.

End users also provide an excellent source of information concerning what data an existing system has available or what data should be made available to a new system. They may not be aware of this knowledge, but a carefully constructed interview and a solid working rapport will ensure that this information will eventually be obtained.

It is important to note that when any new change is introduced in an organization, there will be resistance to its acceptance. This is true whether the change is the introduction of a new employee or the development of a new computer application. In order to improve the chances for success, it is necessary to identify and "bring onboard" the natural leaders within the organization for whom the application is being developed.

2. Functional Requirements

Borrowed from the field of Information Engineering, functional requirements analysis is the process of identifying the steps in the processes that must be performed by the system being developed. The process begins at the lowest level by determining what tasks make up the system. These tasks are divided into "core" and "non-core" functions and are further subdivided to identify greater levels of detail.

In order to identify the core functions for the system, we have to be aware of what requirements must be met for a function to be considered core. We can determine the purpose of the organization, either by examining so-called mission statements or by simply breaking down exactly what the organization does.

Once the requirements for core functionality are outlined, we can identify the individual processes which should be specified as core. These are the lowest level

functions that most closely relate to the purpose of the organization's existence. When first implemented, quick delivery of useful capability is vitally important. The initial core must be viable in the sense that it can operate without future extension while still providing valuable service.

3. Data Requirements

Data requirements analysis is conducted in parallel with functional analysis. It is through the use of data analysis that we determine our basic inputs and outputs. By identifying, in narrative form, the source of our data and the ways that users and management expect output, it is possible to develop a better understanding of what data should be maintained and where it should be located.

a) Identify Data Sources

In analyzing the overall system, it should become apparent where the data originates. Regardless of the type of system, the source of the data will often dictate what type of data serves as an input to the system.

The format of this input may have a very dramatic impact on how the new system deals with the data on the "back end." For most client/server systems this "back end" data repository will be an SQL relational database.

b) Determine Desired Output

The type of information desired from the system can serve as an excellent indicator of what type of data needs to be collected. In new systems, where little or no existing infrastructure exists, identifying the desired output can help determine the type of data that needs to be collected. For example, in order to provide information about an organization's attrition rate, data concerning when and how employees are terminated would be required.

4. Conceptual Database Modeling

The conceptual database model transforms the data requirements narrative, the output of the data requirements analysis phase, into the entities and relationships that form the basis for the system's conceptual schema. In addition, the conceptual database model

provides the planning matrices with the entities necessary to develop the functional mapping, on which the application distribution is based.

a) Define System Entities

Entities are anything in the user's work environment about which he wants to store data. For example, in a sales system, it would be desirable to maintain information on customers, suppliers, and products. By identifying the entities in the system, we develop a high-level view of what we are tracking. This process leads to the natural identification of the data to be maintained by the system. The entities identified in this phase will later be implemented as database tables.

b) Determine Attributes And Relationships

Attributes and relationships define the properties of the entities and how they relate to each other. The attributes of an entity further describe it. Since relationships define how entities relate to each other, they provide some assistance in determining the business rules followed by the system. One must define a relationship between, say, a customer order and line-items on the order. The business rules define how that order should be processed and what should be done in the event there is insufficient stock to fill an order.

B. SYSTEM DESIGN

Once a detailed analysis has been completed, the overall design of the system should be a shared vision of all those involved. In following a construction paradigm, the analysis correlating to architecture, we are now ready to examine the engineering or design phase. Similar to the analysis phase, in design we will address functional and data design.

1. Functional Decomposition

The output of functional analysis, the function list, provides the input to functional design. In the functional design phase, the functions on the list are expanded or decomposed into progressively greater levels of detail. Typically decomposition will go down for at least two levels for purposes of identifying a level of detail that adequately

describes business functions. The output of functional design will serve as the input for the actual Rapid Application Development, or RAD, process.

2. Logical Database Design

In logical database design the conceptual database model developed in the requirements phase is transformed into a database schema. A schema describes the structure of a database. The global logical schema then is the overall structure of the system's databases. To develop the global schema, the component schemas of the application developed during the conceptual phase are integrated into an overall schema.

3. Physical Database Design

Physical database design focuses on the physical design of the actual database tables, the identification of primary and secondary keys, access paths, and other associated data structures. The physical design is specific to the database management system selected for the system. Examples of current popular relational database management systems are Interbase, Oracle, Sybase, Informix, and SQL Server. All these products support the SQL-92 standard, which is the defacto standard for relational databases. Each one of these products also implement special features which are specific to the DBMS. These features include referential integrity, stored procedures, and triggers.

4. Partitioning Design

Unique to the design of client/server systems is the design of distribution architectures. The objective of distribution design is to decide how an application and its data should best be divided. This process will provide the best starting point along logical lines. In order to optimize performance, individual processes must be examined during implementation and be individually "tuned." Tuning is the process of identifying individual processes and determining where that processing should be conducted to maximize performance.

a) Planning Matrices

The first step in distribution design is to determine which entities require specific

functions. This process is most easily accomplished by creating an entity versus function matrix. This matrix displays along the y-axis the functions determined from the previous phase and across the x-axis the entities identified for the system. This provides a visual representation of the logical groupings of functionality.

b) Distribution Design

(1) Conduct Traffic Analysis

Traffic analysis is concerned with identifying the traffic patterns of information flow throughout the process. Unlike work flow analysis, which represents the process of routing and approval that transactions follow in the system, this form of traffic analysis identifies bandwidth requirements for communications links in the system. The focus of traffic analysis is the identification of high volume nodes in the system where substantial communications investment will result in the greatest performance gains.

In implementing the messaging required to automate electronic routing, an examination of the process itself should be conducted. This, however, enters the realm of business process reengineering, which is beyond the scope of this thesis.

(2) Model the client

Developing the model for the client involves determining what data and what functionality logically should be placed on the client. This can be accomplished by inspecting the process/entity cross-tabs and grouping functionality and data that correspond to the same entities.

In some organizations this may be particularly difficult due to political concerns surrounding ownership of data and hardware. This is an area where the support of upper-level management is very important. Senior management would be very effective in breaking down these unnecessary political barriers that can prevent efficient utilization of resources.

(3) Model the server

To correctly model the server several issues must be addressed. These issues

include performance, security, and mission criticality.

(a) Performance and Replication

Replication is the process of maintaining identical copies, or mirrors, of databases in multiple locations. Traditionally, replication is considered undesirable due to the difficulty it introduces in keeping all copies of the data synchronized. It may be desirable, however, to accept this additional complexity in favor of the higher performance that replication sometimes offers.

(b) Security

While it is desired to maintain a seamless connection between a client and a server, security concerns will mandate that the user verify their identity through use of a password. When numerous servers are part of the system, the problem of security can become quite complex. Typically each server will maintain its own security system. Each data source may require its own password. To prevent bombarding the user with numerous logon requests, the application may be responsible for negotiating connections and managing numerous system access requirements. For example, a sales application may access a sales database that maintains clients and a warehouse database that maintains orders and invoices. These databases may be located across the country and require separate security access controls. The application should take care of maintaining the access information for these databases so that the user need provide a password only once.

(c) Mission criticality

If a system can continue to provide its fundamental purpose without a particular process then the process is probably not mission-critical. If a mission-critical process can be identified, then measures to protect that process should be defined. These measures may involve redundant communications lines, contingency power sources, and in extreme cases complete duplicate servers that provide redundant protection from failure.

(4) Develop functional partitions

Because client/server systems are unique, there are a number of possibilities for sharing functionality between the client and server. Partitioning is the act of allocating the activities of the system between the client and the server. Several rules of thumb may be useful in determining this allocation:

- Locate the functionality as close as possible to the source of that functions input and the target of that function's output. Where source and target are at different locations, consider decomposing the functionality further.
- Locate the functionality on the platform that provides the most appropriate resources for the support of that functionality.
- Locate the functionality where it will act to conserve resource in the following priority:
  - Conserve shared server resources.
  - Conserve shared network bandwidth
  - Conserve client resources.[Ref. 6]

(5) Develop repository partitions

Data partitioning is similar in execution to application partitioning. Logically related data may be segregated over a wide geographic region for a number of reasons. Some of the factors that justify the segregation include processing capacity, communications bandwidths, availability, organizational structure, and the combination of heterogeneous data sources.

Conversely, there are many factors that argue against the distribution of data. Some of these factors include:

- Location transparency for application access.
- Performance for distributed queries over wide area networks and relatively slow transmission lines.
- Fully distributed security that encompasses all servers that may provide residence to one or more "chunks" of distributed data.
- Full transaction management, distributed update commit protocols for failure detection and backout, and distributed concurrency control.
- Allowance for failures of localized resources.
- Organizational ownership of distributed data.[Ref. 6]

C. MIDDLEWARE DEVELOPMENT

Connectivity allows applications to transparently communicate with other programs or processes, regardless of their location. The key element of connectivity is the network operating system (NOS). NOS provides services such as routing, distribution, messaging, file and print serving, and network management services. The protocols are divided into three groups: media, transport and client/server protocols. Media protocols determine the type of physical connections used on a network. Types of physical connections include Ethernet, Token Ring, Fiber Distributed Data Interface, and Asynchronous Transfer Mode. A transport protocol provides the mechanism to move packets of data from client to server. Once the physical connection has been established and transport protocols chosen, a client/server protocol is required before the user can access the network services. A client/server protocol dictates the manner in which clients request information and services from a server and also how the server replies to that request. [Ref. 7]

In order to develop system knowledge to an appropriate level to make these decisions, it is necessary to examine the communications plan for the system. This is treated as an independent effort in this methodology because, in a new implementation, the application and data partitions drive the communications requirements.

1. Network Analysis

Depending on the type of migration taking place, a substantial investment in networking infrastructure may be required. In determining the type of network required, it is necessary to determine the area of responsibility and the type of connectivity that the system will require.

a) Determine the Area of Responsibility

Area of responsibility refers to the geographic distance to be spanned by the system. In the case of a departmental system, a single local area network may fulfill the requirement. On the other hand, a university campus, or a single site corporation, could

require segmented LANs utilizing routers. An enterprise wide/global system will require the use of inter-networks and wide area networks. In addition to the physical connectivity, logical connections must be organized in such a way as to optimize performance. For example, a manufacturing company may be physically organized along product lines, but for optimum network performance departmental databases may be desired.

b) Determine System Connectivity

When establishing the overall view of networks for the system, it is necessary to determine how portions of the systems should be connected. There are essentially three types of connectivity options that involve different levels of timeliness. While it is usually thought that real-time updates are best for all parts of the system, this may not indeed be true. Costs of batch transactions may be 10% of real-time systems. Limited resources will dictate that some type of hierarchy be established. Data needs to be timely to the extent of the context in which it is viewed. Having up-to-the-second data in the database is of little use if that data is only used to compile a quarterly, weekly, or even daily report.[Ref. 4]

For example, while one would certainly want real-time access to information concerning transactions in progress, updating sales figures in real-time might cause decision support systems based on them to be too volatile. To establish what aspects of the system should take processed in real-time, it is necessary to examine the criticality, the timeliness, and the level of security of each transaction.

2. Network Design

The system is now ready to be broken into the physical locations that will house the data and the functionality. These locations were identified logically in the partitioning design phase by cross referencing the entities to the functions they perform.

a) Identify repositories

The combination of server models, data partitions, and traffic/workflow analysis help identify the location of data repositories. Repositories are logical locations to place the physical databases. By identifying the physical locations that house the data and

functionality, it becomes obvious which locations must be connected.

b) Determine bandwidth requirements

Bandwidth requirements are directly related to the volume of traffic expected both between clients and servers and between multiple servers. Bandwidth refers to the amount of data that can be moved thorough a communications channel in a given amount of time. Typical bandwidth measurements are thousands and millions of bits per second (kbps/Mbps).

c) Identify replication candidates

Data repositories that should be replicated will not be immediately apparent. It is not uncommon for replication to occur in the second or third wave of optimization. As stated before, replication should be a last resort for optimization. The difficulty involved in maintaining accurate copies of multiple databases far outweighs most of the performance benefits.

3. Network Installation/Improvement

Network installation and improvement may require significant construction to existing physical facilities, depending on the degree of installation being considered. If possible, the computer network infrastructure should be planned when a building is constructed. Installation of network cabling and hardware can be very costly and intrusive when conducted after a building is completed.

D. SYSTEM IMPLEMENTATION

The implementation of the system is the first part of the process that provides the client with something tangible that they can respond to. The development and use of Rapid Application Development techniques is key to keeping the client involved in the process. Under traditional methodologies it was not uncommon to analyze and design a system and deliver the finished product two or three years later. It is obvious that in rapidly shifting business environments, this is an unacceptable turn-around. The implementation proposed here focuses on RAD, automated database development and

modern communication networks to quickly provide the client with an acceptable solution.

1. RAD Cycle

Rapid Application Development, or RAD, is quickly becoming the most popular method for delivering client/server applications. The reason for this popularity is that RAD focuses on delivering quality software as quickly as possible. Contrary to the traditional *waterfall* model, which depicts software development as a sequential series of processes, RAD views development as an iterative process. Prototyping, testing, review, and improvement are repeated frequently throughout the development process. RAD focuses on developing software incrementally. Fred Brooks, in the classic *Mythical Man Month*, recommends "...build one to throw away." [Ref. 8] In RAD, the application is built many times, continually improving on what has already been built.

a) Prototype

The prototype is central in the rapid development of applications. The process of prototyping involves developers and end users working closer than previous approaches suggest. Developers gain a sense of reality and urgency from working with end users, and user requests are easily conveyed to the developers because of the absence of multiple layers of administrative overhead.

Prototyping focuses on delivering working applications, or application parts, quickly so that ideas and conceptual issues can be discussed early in the development process. Event driven programming and the graphical user interface promote the prototyping approach, by allowing developers to quickly generate applications that the user can interact with and provide useful feedback.

(1) Event model

Traditional PC-based applications are based on a sequential model where the basic *flow* of the application is decided by the application developer at design time. Modern operating systems allow the use of multitasking and windowing environments to provide the user with the luxury of controlling the execution of the application at run time. This

ability is referred to as the event model. In an *event* driven application, the user decides what will take place and when it will occur. Some structure, of course, still exists, but the processing of the application is based on responding to different user-initiated events.

(2) Graphical user interface

Graphical user interfaces, or GUIs, allow users to interact with a computer in a more intuitive way. A GUI consists of standard visual components like buttons, drop-down lists, radio buttons, and check boxes that provide a consistent visual representation of the application. GUIs help make using application programs appear as a natural extension of the way an individual works.

(3) Implementation tool

There are numerous tools available to improve the process of prototyping applications. These products range from *screen painters* that present static images which will later be translated into operational code, to *application generators* that eliminate the need for the developer to write the code for the user interface. Powerful application generators have recently appeared on the market that allow the developer to interactively manipulate the GUI elements and have the code modified in real time.

b) Test

Testing occurs frequently and interactively in RAD. Because development occurs in such rapid succession, the ability to frequently and interactively test the application is extremely important. Modern tools typically provide a run operation that allow the developer to try ideas interactively, eliminating the high cost of long time periods typically associated with assembling an application.

c) Review

Review provides application developers with the input they need to develop intuitive applications that respond to the needs of their users. Review should be frequent and involve developers as well as end users. This type of review provides users and developers with feedback throughout the development process so no surprises occur when

the system is delivered.

During the review process, an application developer should describe the function and structure of the code that has been developed. It is often most advantageous to have a developer perform the presentation who did not actually develop the code being described. This approach would inspire developers to write code that is easier to understand and explain.

d) Improve

Continuous improvement is a philosophy that has not always been a part of traditional development approaches. Using RAD, developers are more involved with the design of applications, and are therefore more likely to build in quality. Knowing that the code they are developing will be revisited, inspires more intelligible comments and better design. Most information systems organizations are trying to be more proactive in their strategic planning. "Because IS [departments are] less likely to be mired under an incredible backlog of change requests (end users will be able to do a lot more on their own) and because staffs are likely to know more about the company's business, IS [departments] can actually anticipate user's requests." [Ref. 1]

2. Database Implementation

If a procedural approach has been followed in designing the database, the transition from logical schema to physical tables is a simple one. If utilizing an automated tool, the transition may be even simpler. Typically automated tools will generate tables, primary, secondary and foreign keys, and provide referential integrity. Different tools implement this functionality differently. Some generate an SQL *script* that can then be loaded into a database management system and compiled to produce the physical implementation. Other tools will actually imbed this function in the product itself and generate the tables automatically.

E. SYSTEM MAINTENANCE AND EVOLUTION

1. Application Maintenance

Because an application developed with rapid application development techniques is in a continual state of improvement, the actual maintenance phase is more difficult to define. One could argue that application maintenance begins once the initial working prototype has been delivered. Alternately, because an application developed using RAD techniques continually "evolves," one could argue that application maintenance never actually begins.

2. Data Maintenance

Most data maintenance tasks have been automated by utilities that ease the workload of a database administrator. Even so, an individual within the organization should be appointed to act as the database administrator to manage user access and system configuration. This individual will be responsible for maintaining user rights and controlling the access permissions for certain groups of users. Depending on the size of the system, this duty may be a part-time/collateral duty. In larger systems a full-time position should be considered.

3. Communications System Maintenance

Communications system maintenance covers a broad area of responsibility and involves a number of potential parties. Wide area network maintenance and improvement will almost always be the concern of the service provider. Local area network maintenance may be a task handled internally or may be outsourced depending on the level of control desired.

In this chapter we have examined a modern methodology for analyzing, designing, developing, and evolving client/server applications. Chapter IV discusses the analysis, design, and implementation of a practical application based on the methodology presented in this chapter.

IV. HUMAN RESOURCES OFFICE - A CASE STUDY

A. INTRODUCTION

The Human Resources Office (HRO) of the Naval Postgraduate School (NPS) maintains data on all civilian personnel employed at NPS in accordance with regulations defined by the Office of Personnel Management (OPM). Primarily HRO is interested in maintaining all information about past and present civilian employees. This includes, but is not limited to, personal information, job history information, and payroll information. The information maintained by HRO is considered sensitive since it contains private, personal, and pay and earnings information. In addition to maintaining information on all civilian employees at NPS, HRO is also responsible for maintaining information for every position at NPS whether vacant or filled.

As with most personnel systems, automated or manual, the vast majority of data in the system pertains to individuals data. In order to ensure that all concerned parties have access to the data, a system which provides a central repository for all civilian personnel information is maintained by the Department of Defense. The information in this system relates directly to other systems, specifically, pay and retirement systems.

B. DEFENSE CIVILIAN PERSONNEL DATA SYSTEM (DCPDS)

The Defense Civilian Personnel Data System (DCPDS) is the system currently employed by HRO for the maintenance of information about personnel and positions at NPS. The DCPDS has been in existence since 1982 and has been used to provide an automated system for the storage, transaction management, and retrieval of personnel actions and data.

1. Components

The DCPDS, is a nationwide, mainframe based, information system located in San Antonio, Texas. The system is implemented on a Unisys OS 1100, a multi-user, multi-processor time-sharing mainframe system.

Interaction with the computer system is accomplished via dumb terminals or personal computers running terminal emulation software that are connected via leased telephone lines to the central processor in Austin, Texas.

DCPDS is a data management application written in COBOL, a third generation computer language which stores its data in flat files. As such, it suffers from the problems associated with file processing systems. These problems include data duplication, application program dependency, and difficulty of representing the users view of data.

2. Functionality

DCPDS was designed to improve the accuracy and response time of civilian personnel information required for personnel management at activity and headquarters levels. It was also designed to provide a uniform system for civilian personnel systems DoD-wide.

DCPDS eliminates some, and reduces other, activity level clerical routines. It provides immediate access for civilian personnel data update and inquiry, documents personnel actions, and maintains historical data that is used for planning, analysis, and reporting.

Users interact with the system via a scripting language that was developed as a part of the DCPDS application. This scripting language makes use of key/command words and data identification numbers. Data identification numbers, or DINs, are alphanumeric designation which describe the data and indicate where it is stored in the DCPDS. Figure 4 shows the overall architecture of the system.

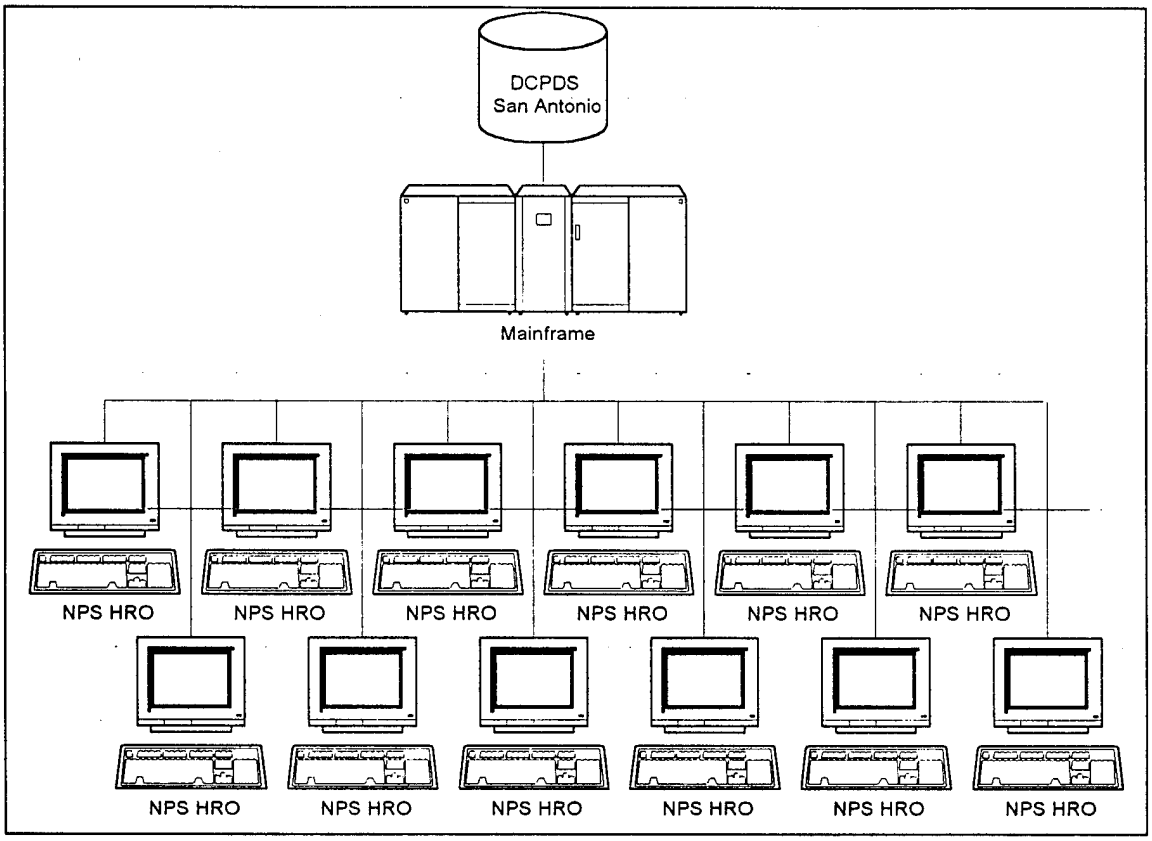


Figure 4: Defense Civilian Personnel Data System

3. Limitations

As indicated earlier, most COBOL systems maintain data in flat files. A flat file maintains a single record for each entity in the system. One of the obvious limitations of this type of data store is that it is necessary to know in advance how many occurrences there will be of each data element. For example, in Table 1 we can observe that there are five occurrences of award dates. What would occur if Mr. Willis were to receive an additional award. In addition, as is obvious from the table, Mr. Willis is the only individual with more than one award. Regardless of that fact, we must reserve room in the data store for all awards that any employee might receive. This is clearly an inefficient use of secondary storage.

Name	SSN	Position	Award #1	Award #2	Award #3	Award #4	Award #5
Willis	123456789	Builder	2/1/69	6/5/71	4/2/75	1/9/86	9/4/93
Cruz	234567890	Fireman	5/7/94				
Pitt	345678901	Security	4/8/90				
Hanks	456789012	Teacher					

Table 1
An Example Flat File

In the early 1970's a researcher with IBM named E.F. Codd developed the *Relational Database Model*.[Ref. 8] The relational model overcame many of the limitations of flat file systems. Relational models *normalize* the data by breaking it into several files (tables) in order to eliminate redundant data and isolate repeating groups of data. This results in multiple tables which can be related to each other through common data elements. Returning to our example, a relational representation might look like Tables 2a and 2b. Even in this oversimplified example, it is clear that Table 1 requires 32 fields, or cells, and that the combined storage of Tables 2a and 2b are only 26 fields.

Name	SSN	Position
Willis	123456789	Builder
Cruz	234567890	Fireman
Pitt	345678901	Security
Hanks	456789012	Teacher

Table 2a
Example Employee Table

SSN	Award Date
123456789	2/1/69
123456789	6/5/71
123456789	4/2/75
123456789	1/9/86
123456789	9/4/93
234567890	5/7/94
345678901	4/8/90

Table 2b
Example Employee Award Table

A second drawback of using the DCPDS is the communications overhead. Since the DCPDS relies on dumb terminals that are connected to it directly, a communications line must be kept open for each individual user, and the number of users is limited by the number of available connections, currently 12 for NPS.

Additionally, DCPDS utilizes its own *command code* language for interaction with the system. This code is a cryptic set of commands and data identifiers that outline the possible personnel actions and requests for information, referred to as *desires*. Because of the level of training necessary to utilize the capabilities of DCPDS, the only personnel who could interact with the system are trained personnel specialists.

Personnel actions are generated throughout the command on standard DoD form SF-52. These forms are routed by hand for approval before being routed to the HRO. In the HRO the forms are processed by teams of three individuals who convert the information into DCPDS code and enter that code into DCPDS. This entire process is manpower intensive and can result in significant delays, and mishandled personnel actions.

Because the information contained in DCPDS is difficult to withdraw and few organizations within the command are aware of its existence, the information is under-utilized. Departments who are forced to provide input to this system are unable to receive information back from the system. To compensate for the perceived void of information, systems have been devised within organizations at the command that duplicate data already contained in DCPDS. These systems were devised to provide a service for which a demand exists, and validate the premise that users will put this type of information to use if it were available.

```
PERS RECD CA CCPO QH STATUS 16.
FM AREA1 TO BE 1:36:X NM1 TO BE 1:18:X DP1 TO 19:8:X TP1 TO BE
    27:10:X CM1 TO BE 27:3:X AV1 TI BE 30:3:X EX1 TO BE 33:4:X
COPY ORG-CD2*26
DC AA TO BE IF BBA<1X26#> WITHIN "A" THRU "G".
DC BB TO BE IF BBA<1X26#> WITHIN "H" THRU "O".
DC CX TO BE IF BBA<1X26#> WITHIN "P" THRU "Z".
```

Figure 5 : Example DCPDS Script

Because of deficiencies previously mentioned, some personnel offices resorted to developing their own micro-computer based systems for local use. One such system that has subsequently been developed for widespread distribution and use is the Electronic System For Personnel.

C. ELECTRONIC SYSTEM FOR PERSONNEL (ESP)

The Electronic System For Personnel, or ESP, grew out of an individual activity's frustration for working with DCPDS. The system is centered around IBM-PC compatible micro-computers that have become more prevalent. ESP automates the *generation* of personnel actions. In contrast to DCPDS which automates the processing of personnel actions, ESP is focused on automating the entire process, from initial draft of the personnel action, through approval and processing, and into DCPDS. In addition, ESP provides some limited local reporting capability and minimal local control of data. ESP is written in Microsoft Fox-Pro for MS-DOS, a popular database package based on the dBase family originally developed by Ashton-Tate Corporation.

The user interface for ESP is a based on procedural menus and forms that mimic the structure and content of the SF-52. This paradigm smoothes the transition from paper based forms to electronic forms and provides a consistent *feel* for users who have been accustomed to using a paper driven system. These menus walk the user through a pre-planned series of steps requiring information in specific order to process requests and transactions.

As previously stated, ESP automates the generation and routing of SF-52's and acts as a front end for DCPDS. Instead of filling out an SF-52 manually, ESP allows offices throughout the command to generate electronic SF-52's which are then routed for approval via a computer network. The approved personnel actions are then routed to the HRO for review and quality control before being uploaded to the DCPDS. The goal of the ESP is the *paperless personnel office*. While ESP goes a long way to improving the paper driven, centrally controlled interface previously utilized by DCPDS, it neglects several key

issues of functionality.

While procedural menus and electronic SF-52's are far superior to the command line language required by DCPDS, they offer little adaptability and are less useable than the event driven system most computer users have come to expect from modern applications such as those developed for use with Microsoft Windows.

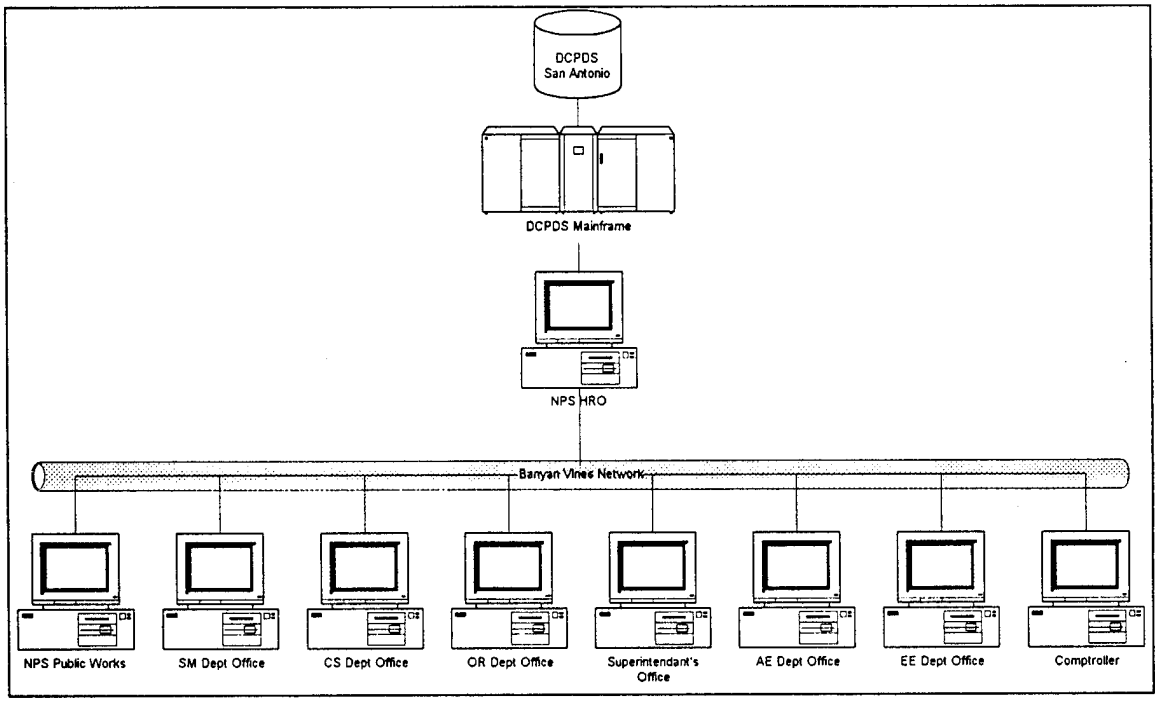


Figure 6 : Electronic System for Personnel

ESP does maintain some of the data it uses in a set of local database tables, but these tables are used mainly for help screens and system tables. One of the tables ESP does maintain is a subset of the DCPDS table 2. Table 2 maintains the master list of all data identification numbers and the system information about them. ESP relies on a subset of table 2 that ships with the ESP application. When updates occur to table 2, which can be as frequent as quarterly, ESP must be modified and redistributed. Unacceptable system downtimes are a direct result of this dependency.

There is little in the way of reporting that ESP can perform on its own. For most reporting information, the personnel specialist must revert to DCPDS language scripts.

Neither ESP nor DCPDS offer any ability to generate summary reports. All summary reports must be compiled by hand.

Because ESP functions, essentially, as a front end to DCPDS it maintains many of the requirements and limitations that are inherent in DCPDS. This means that the number of concurrent links to DCPDS is still limited to twelve. While many more users can generate personnel actions, desires and reports can only be entered in the system via the twelve systems connected directly to DCPDS.

D. CLIENT/SERVER PROVIDES THE ANSWER

ESP goes a long way to improving the automation of personnel systems for DoD. There are, however, many features that a modern system could provide that would aide usability and improve functionality.

Because such a large investment already exists in the central DCPDS system, it is undesirable, if not impossible, to change that system in the short term. The solution proposed here alleviates demand on that system, provides users with a user friendly intuitive system, with the power to create information quickly and easily.

1. Desired Traits of a Proposed Solution

The central DCPDS mainframe computer maintains connections with numerous other systems to maintain current information for such systems as payroll and retirement. In order to preserve the overall system and the complex interfaces it maintains with these other higher level systems, it is desirable, if not mandatory, that the appearance of the system from the DCPDS side remain unchanged.

The system must be user friendly, allowing people with little or no training to utilize the system for data input and information retrieval. In the past users have used paper forms and electronic forms that mimic those paper forms. This system must provide a paradigm for these users that is at least as easy to understand. Ideally, the user interface should allow the user to customize the interface to his or her personal preferences, while

maintaining some overall application structure.

The most glaring deficiency in both DCDPS and ESP is the lack of reporting capability. While the automation of personnel action generation and processing is highly desirable, the advantage to having an automated system is that it permits management to draw *information* from the *data* that it collects. While detail reports fill a necessary requirement, the true power of an information system is its ability to display summary data that management can use to identify trends and potential problems. Because DCPDS and ESP have both proven to be weak in this area, it is an area that has received particular emphasis in this application.

DCPDS is a central system. The managing agency in San Antonio decides when the system is available and how the system can be loaded. Because of differences in time zones and other factors, the system may be highly loaded before the local office can even connect to the system. This can introduce long, variable delays that prevent the utilization of the central system. The solution is to allow the local office to have full control over their data. The local office is held accountable for the data in accordance with regulations. Since the local office is accountable for the data, they should be allowed to have full control of it.

2. Impetus for Change

Recognizing some of these deficiencies and in an effort to increase customer satisfaction and reduce costs, a dialog was initiated between the director of the Human Resources Office (HRO), Ms. Mary Aguilar, and Dr. James Emery of the Systems Management department. As a process of continuous improvement, Ms. Aguilar has envisioned providing NPS and possibly other DoD commands with a system which facilitates rapid access to information, real time manipulation of data, and flexible report generation, all at a cost savings to the government.

3. Human Resources System - The Proposal

The solution envisioned for this system is unique in that it maintains the central database with absolutely no modification to that system. What is proposed is a client/server system maintained locally that will retrieve, on a periodic basis, a snapshot of the data that pertains to the command. Initially, this snapshot would occur on a frequent basis (perhaps as often as once per day) to ensure quality. After the system is fully tested and confidence established in the system, the snapshot can be scheduled less frequently. The data snapshot

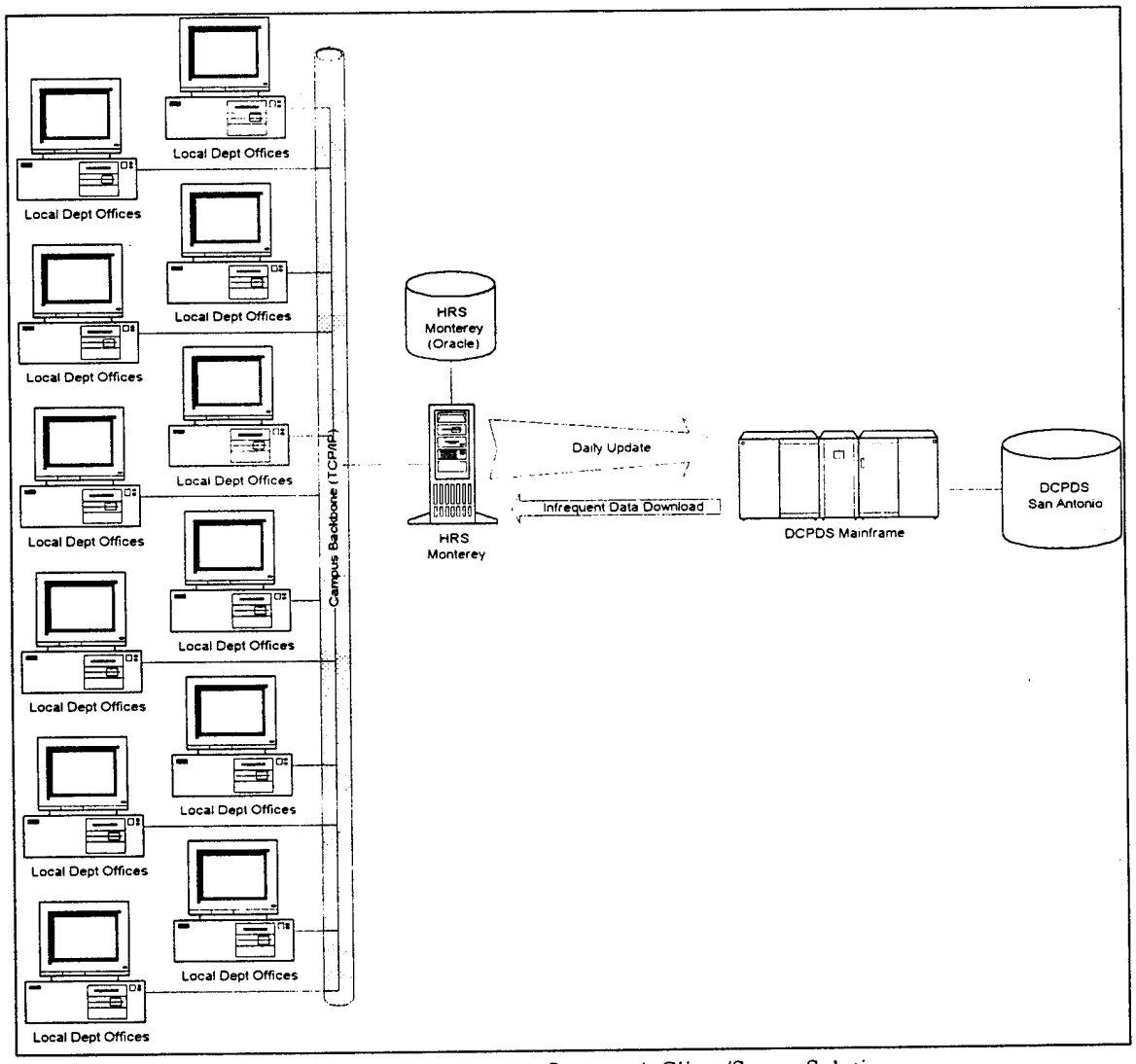


Figure 7 : Human Resources System, A Client/Server Solution.

is automatically converted from the flat text files, received from the DCPDS, to relational tables maintained on a local SQL relational database server. Once the data is accessible on the local server, transactions can be executed and reports generated locally. Figure 7 shows the overall architecture of this proposal.

a) Functionality

The transactions that take place during the day are logged, executed against the local data in real time, and converted to an equivalent DCPDS transaction. These DCPDS transactions are stored locally until approximately one hour prior to the time DCPDS goes off line for the day. In a single communication transaction the entire day's transactions are uploaded to the DCPDS where they will be processed in batch fashion while the DCPDS is off-line.

HRS, like the ESP, relies on the DCPDS table 2 for mapping data elements in DCPDS to the local system. Unlike ESP, HRS is capable of automatically recognizing that modifications to DCPDS have occurred. HRS will immediately request an updated copy of table 2 from the DCPDS. The system administrator at the command level can then determine the appropriate mapping for any new data elements, remove data elements that are no longer used, or modify data elements that have changed. This feature improves system availability and provides the system administrator with the power to customize the local system.

b) Development Tools

The user interface for this application has been designed for Microsoft Windows based IBM compatible personal computers. The application itself was constructed utilizing a new and very powerful Rapid Application Development tool from Borland International, named Delphi. It provides a unique set of powerful tools and a liberal distribution license for any applications developed using it.

Delphi incorporates a combination of features that make developing applications with a RAD philosophy easier and more predictable. The Delphi environment boasts an

with a RAD philosophy easier and more predictable. The Delphi environment boasts an object oriented language, a native optimizing code compiler, intuitive visual tools, and sophisticated database tools.

Object Pascal provides the core technology for the Delphi environment. The compiler that Delphi uses is a mature product that has been utilized extensively in industry over the last 10 years. Delphi compiles at a rate of over 300,000 lines of code per minute on an Intel ™ Pentium ™ based personal computer. Object Pascal is an object oriented language that aggressively promotes code reuse through the use of visual components that encapsulate standardized functionality. The functionality may be extended either by inheriting new objects from those existing or by writing event handlers to respond to certain events that the object may experience.

Delphi's visual tools reduce the generation of the user interface to a simplistic task. This allows the software developer to quickly develop the user interface and move on to the generation of the actual application logic.

Packaged with the client/server version of Delphi are several tools that simplify the process of utilizing remote database servers. SQL Links provide seamless integration of data from Oracle, Sybase, SQL Server, and Interbase. This middleware product is provided and may be redistributed with any application developed with Delphi completely free of royalties. Additionally, the Visual Query Builder and ReportSmith are included to ease the task of developing SQL queries and generating complex reports.

c) Costs and Benefits

As with any system, there are costs and benefits associated with an implementation. These must be carefully weighed to determine if the benefits are worth the costs and that the risks are low enough to make the implementation successful.

The most obvious benefit of the system proposed is the additional functionality and ease of use that can be derived from the system. Higher productivity will translate to savings in personnel budgets, either from lower rates of attrition or from decreases in

telephone lines on-line for ten hours per day five days a week or approximately 144 communication hours. The solution proposed here would require a single line for less than one-half hour each day or one-half of one communication hour. Communications charges are essentially eliminated.

The cost of developing the application software is significantly less than that experienced in a system such as ESP. Conservative estimates suggest a full time equivalent manning level of eight developers working on ESP over a period of four years. This represents an investment of 32 person years. Generous estimates for HRS indicate total development costs of two developers for six months or 1 person year. This represents a significant savings just in development personnel costs.

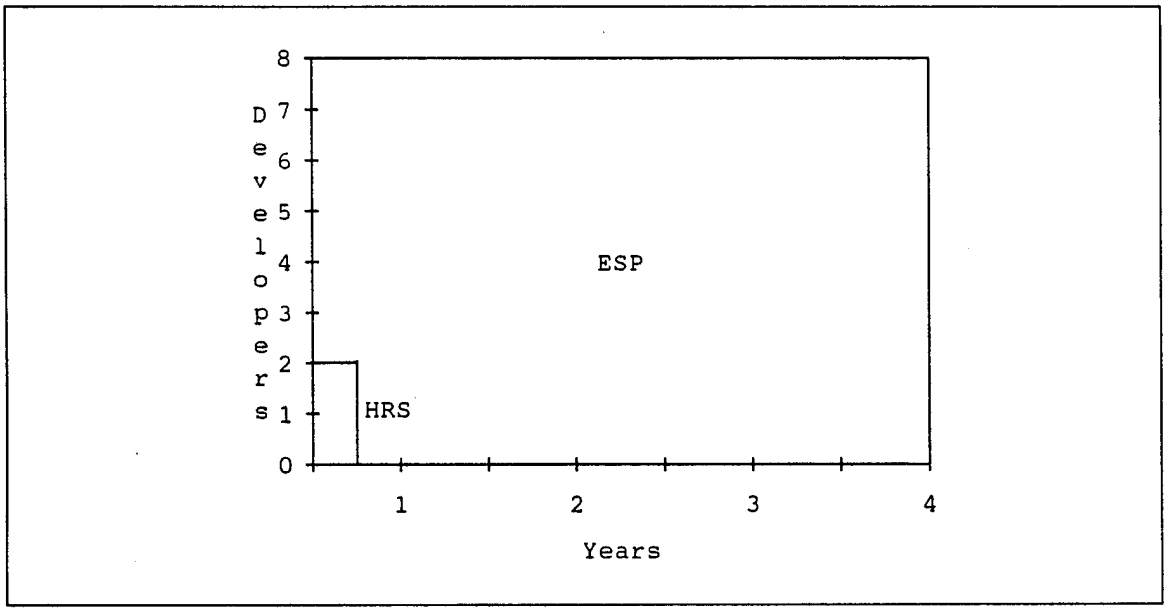


Figure 8
Comparison of ESP & HRS Development Costs in Person Years

Because applications developed with Borland Delphi can be distributed completely free of royalties, there would be no additional cost incurred for application software. The only other significant cost would be that of an SQL server for each command that chooses to implement the system. Were a compatible SQL server already available at the

only other significant cost would be that of an SQL server for each command that chooses to implement the system. Were a compatible SQL server already available at the command, this cost could be forgone.

HRS provides an excellent model for providing a bridge from traditional DoD legacy systems to modern client/server systems. It does so without requiring changes to large central systems with complex interfaces. Additionally, it maintains an existing investment in infrastructure until such time as those entire central systems can be modernized.

This chapter has described the implementation of a client/server architecture to a DoD legacy system. The solution set is unique because it maintains the legacy mainframe system to provide stable connectivity with other DoD systems. The application described provides a model for the migration of DoD legacy systems to client/server architectures. Chapter V discusses the findings generated by this thesis and the recommendations for future research that will further foster the use of client/server architectures and RAD techniques for DoD.

V. LESSONS LEARNED AND FURTHER RESEARCH

This chapter concludes the thesis by presenting the lessons learned during the course of developing the thesis and the associated application. The chapter also presents directions for future research.

A. LESSONS LEARNED

1. Small to medium sized system development is well suited for RAD methodology

The most significant lesson learned is that formal, structured methodologies can detract from the accomplishment of productive work for small to medium size systems. A rapid application development methodology based on data modeling and prototyping seems to be a more appropriate approach for developing such systems. The iterative process of a RAD methodology and its user orientation result in systems that meet user requirements and are more likely to be accepted and adopted by users. It is still expected that an application of significant complexity would require a formal methodology.

2. Modern application development tools greatly enhance the development process.

The new generation of application development tools based on event-driven, object oriented, visual components and advanced compiler technology greatly enhances the productivity for developing applications. They are also well suited for the RAD philosophy, since changes can be incorporated easily.

3. Upper level management and end-user support and enthusiasm are indispensable for the success of application development projects

The support and enthusiasm of management and end-users proved to be a key factor in the success of this application. Continual interest by management in this project

provided focus and helped overcome any obstacles to its completion. It is hard to imagine undertaking big application development projects without the full support of top management and the involvement of end-users.

4. Data is the key for the success of any system

With modern application development tools, implementing required functionality is becoming a relatively easy task as long as the data it requires is available. An advantage of client/server technology is its ability to access data where it resides, regardless of the platform type on which it resides. The ability to identify data requirements and locations is, therefore, a crucial requirement for the success of any system.

5. Complex applications would most likely benefit from a conceptual database modeling tool.

When developing small or medium size applications a conceptual data model may be considered optional. It is very likely that conceptual data modeling should be considered mandatory for complex systems. The complexity of the system will necessitate the capture of data requirements separate from implementation details. This activity will require the use of an automated tool to develop the conceptual model and possibly generate the implementation database schema. Candidate conceptual data model include the Entity Relationship Model, IDEF1X, and Object Oriented Data Modeling. In developing the application of this thesis, a conceptual modeling tool was employed in the early stages of database design, but proved useless for such a simplistic data model.

6. Communications interfaces are the most difficult to implement.

By far, the most difficult aspect of developing this application was the identification and analysis of the data communications requirements. The mainframe system that runs DCPDS utilizes an obscure communications protocol which posed many difficulties in the implementation of the update mechanism. It is suspected that connectivity and data communications issues between clients and servers through

middleware will also be a difficult aspect of client/server development.

7. **Client/server technology allows the peaceful coexistence of modern and legacy systems**

Because of the separation of application environments from accessed data inherent in client/server approaches, sophisticated client applications can be built with modern tools and yet be able to access legacy data. These systems can provide capabilities that can not possibly be obtained from legacy systems and at a fraction of the cost and time required to develop their predecessors.

8. **Providing unfulfilled functionality increases acceptance of the system**

By placing emphasis on services not previously provided, user acceptance and support can be maintained at a high level. This approach allows the introduction of new technology with less resistance. Once the user is accustomed to the newly introduced technology, additional functionality may be incorporated.

B. FUTURE WORK

The following are enhancements to the developed system:

1. **Functional enhancements**

a) Updates

Because ESP was already in the process of being implemented throughout the command, a decision was made that shifted the focus of HRS away from the initial design. Instead of immediately attempting to alter the automated routing of personnel actions, the focus was shifted to providing reporting capabilities absent from both ESP and DCPDS. For this reason, work on the data editor form and its associated functional code was halted to provide adequate time to complete the reporting capabilities of the system.

b) Electronic Messaging and Routing

Because the focus of HRS was directed away from the automation of personnel

actions, the electronic messaging and routing module of the application was not developed. It is necessary to develop this capability if HRS is to eventually replace ESP.

c) Data Communications Protocols

The data communications protocols that would allow direct connection to the mainframe in San Antonio could not be procured and implemented in the time allotted for this research. In order to provide a seamless integration of HRS with DCPDS, this functionality must be added to the system.

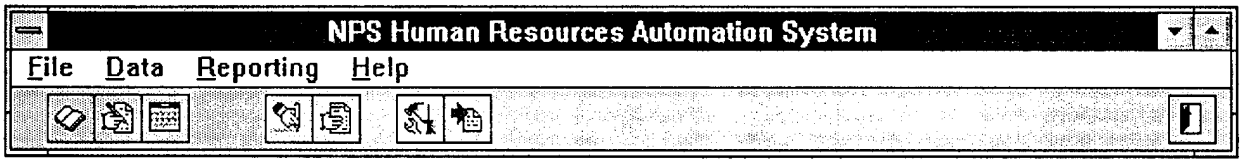
2. User Interface Enhancements

Throughout the on-going development of this system, the user interface should be continuously revisited to ensure suitability to task. Every effort should be made to enable the user to modify the interface to best accommodate individual work preferences. It is worth noting that developers utilizing RAD tools such as Borland Delphi are much more conscious of interface concerns because they must continuously view and manipulate the interface itself, not a computer code representation of the interface.

3. Performance Issues

When the system is migrated to its eventual implementation on an Oracle database server and placed in operation, procedures that consume excess network bandwidth or client resources should be identified for possible implementation as stored procedures or triggers. As experience with the system grows, the most common sorting patterns for data will become familiar to the developers. At such time, indices should be created to enhance responsiveness and increase overall system performance.

APPENDIX. HRS APPLICATION FORMS AND SOURCE CODE

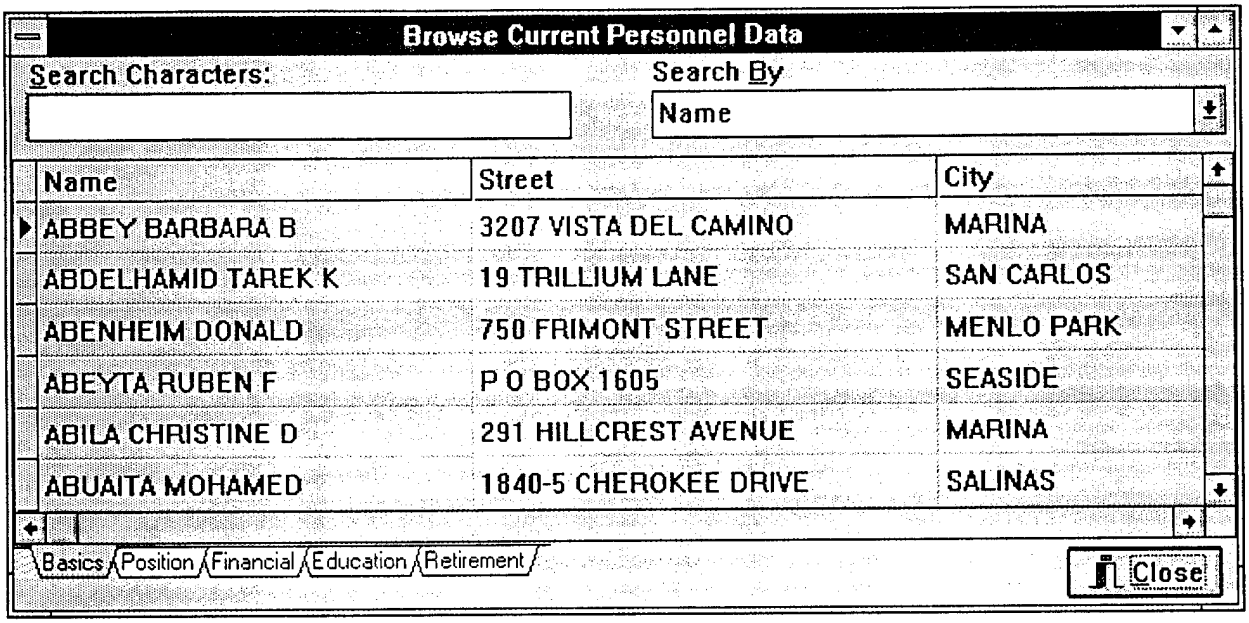


File Name: UHRODEMO.PAS

```
unit Uhrodemo;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,  Controls,
  Forms, Dialogs, Buttons, Menus, ExtCtrls, UBrowse, UTestOne,
  UDesires, UAbout, UEditor, Report;

type
  TForm1 = class(TForm)
    Panel1: TPanel;
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    Exit1: TMenuItem;
    N1: TMenuItem;
    PrintSetup1: TMenuItem;
    Help1: TMenuItem;
    About1: TMenuItem;
    HowtoUseHelp1: TMenuItem;
    SearchforHelpOn1: TMenuItem;
    Contents1: TMenuItem;
    Data1: TMenuItem;
    btnCurrent: TSpeedButton;
    btnHistory: TSpeedButton;
    btnExit: TSpeedButton;
    btnReports: TSpeedButton;
    btnMaint: TSpeedButton;
    btnDesires: TSpeedButton;
    btnAdHoc: TSpeedButton;
    Reporting1: TMenuItem;
    Report1: TReport;
    BrowseCurrentData1: TMenuItem;
    BrowseHistoryData1: TMenuItem;
    PreparedReports1: TMenuItem;
    CreateNewReports1: TMenuItem;
    PtrSetupDlg: TPrinterSetupDialog;
    OpenDialog1: TOpenDialog;
    SpeedButton1: TSpeedButton;
    procedure btnCurrentClick(Sender: TObject);
    procedure btnExitClick(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
```

```
    procedure btnMaintClick(Sender: TObject);
    procedure btnReportsClick(Sender: TObject);
    procedure btnDesiresClick(Sender: TObject);
    procedure btnAdHocClick(Sender: TObject);
    procedure PrintSetup1Click(Sender: TObject);
    procedure btnHistoryClick(Sender: TObject);
    procedure About1Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.btnCurrentClick(Sender: TObject);
begin
  Form2.Show;
end;

procedure TForm1.btnExitClick(Sender: TObject);
begin
  Close;
end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Form2.Hide;
  Form2.Free;
end;

procedure TForm1.btnMaintClick(Sender: TObject);
begin
  frmMaintenance.Show;
end;

procedure TForm1.btnReportsClick(Sender: TObject);
begin
  if OpenDialog1.Execute then
  begin
    Report1.ReportDir := ExtractFilePath(OpenDialog1.Filename);
    Report1.ReportName := ExtractFileName(OpenDialog1.Filename);
    Report1.Run;
  end;
end;
```

```
procedure TForm1.btnDesiresClick(Sender: TObject);
begin
  frmDesire.Show;
end;

procedure TForm1.btnAdHocClick(Sender: TObject);
var
  CmdStr : PChar;
begin
  Form1.Cursor := crHourGlass;
  CmdStr := 'c:\rptsmith\rptsmith.exe';
  if WinExec(CmdStr,SW_SHOWNORMAL) < 32 then
    MessageDlg(('Could not execute Report Smith.'+Chr(13)+
      '        An error has occurred'),mtError,[mbOk],0)
  else
    Form1.Cursor := crDefault;
end;

procedure TForm1.PrintSetup1Click(Sender: TObject);
begin
  PtrSetupDlg.Execute;
end;

procedure TForm1.btnHistoryClick(Sender: TObject);
begin
  MessageDlg('Feature Not Yet Implemented', mtInformation, [mbOK], 0);
end;

procedure TForm1.About1Click(Sender: TObject);
var
  frmAbout : TfrmAbout;
begin
  frmAbout := TfrmAbout.Create(Self);
  try
    frmAbout.ShowModal;
  finally
    frmAbout.Hide;
    frmAbout.Free;
  end;
end;

procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
  frmEditor.Show;
end;

end.
```



File Name: UBROWSE.PAS

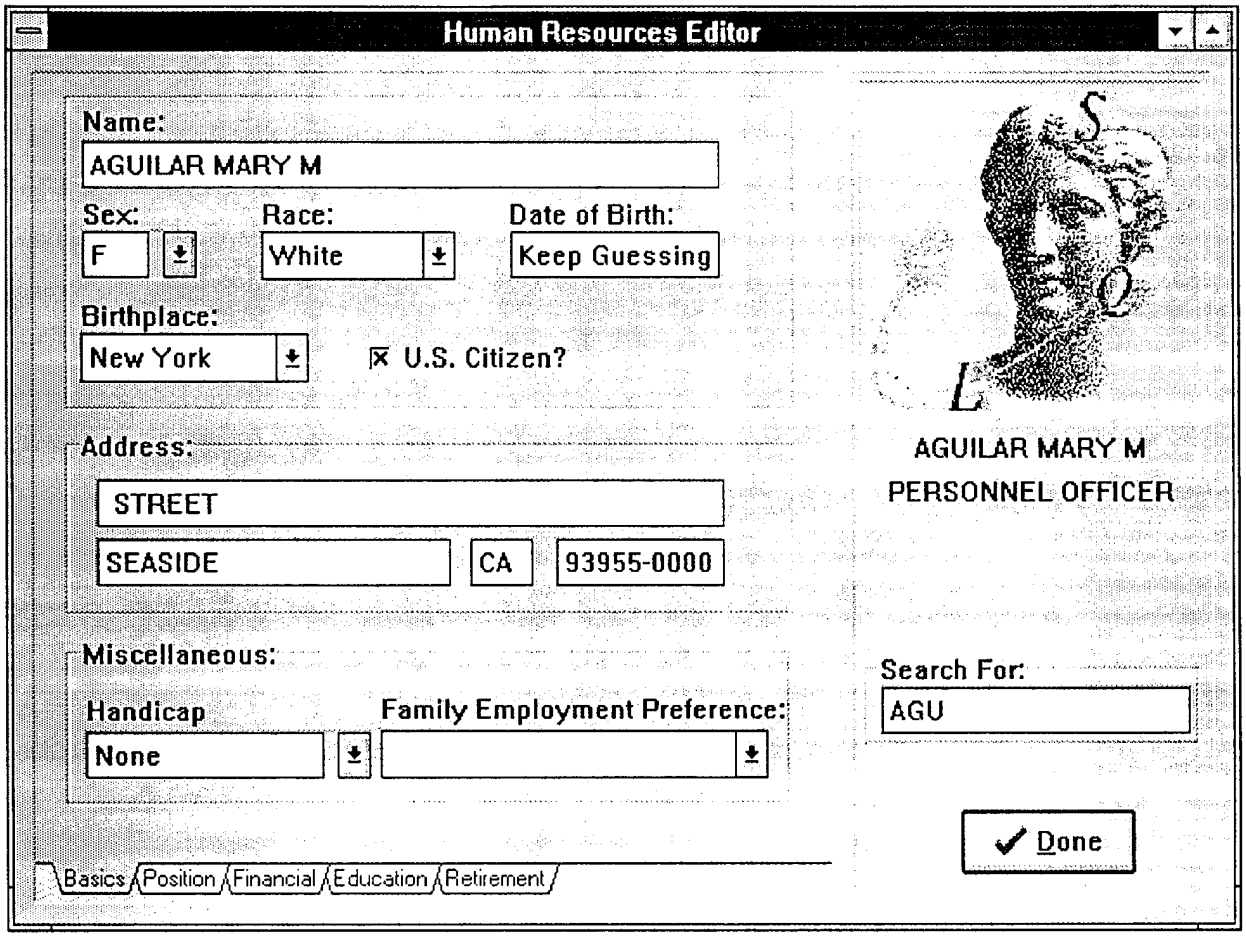
```
unit Ubrowse;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons, Wwkeycb, Grids, Wwdbigrd, Wwdbgrid,
  DB, DBTables, Wwtable, Wwdatsrc, Tabs, wwdblook, ExtCtrls, Wwdotdot,
  KingPop;

type
  TForm2 = class(TForm)
    wwDataSource1: TwwDataSource;
    wwTable1: TwwTable;
    BitBtn1: TBitBtn;
    TabSet1: TTabSet;
    wwTable6: TwwTable;
    wwDataSource6: TwwDataSource;
    Notebook1: TNotebook;
    wwDBGrid1: TwwDBGrid;
    wwDBLookupCombo1: TwwDBLookupCombo;
    wwIncrementalSearch1: TwwIncrementalSearch;
    wwKeyCombo1: TwwKeyCombo;
    Label2: TLabel;
    Label1: TLabel;
    wwDBGrid2: TwwDBGrid;
    wwDBLookupCombo2: TwwDBLookupCombo;
    wwIncrementalSearch2: TwwIncrementalSearch;
    wwKeyCombo2: TwwKeyCombo;
    Label3: TLabel;
    Label4: TLabel;
    wwDBGrid3: TwwDBGrid;
```

```
wwDBLookupCombo3: TwwDBLookupCombo;
wwIncrementalSearch3: TwwIncrementalSearch;
wwKeyCombo3: TwwKeyCombo;
Label5: TLabel;
Label6: TLabel;
wwDBGrid4: TwwDBGrid;
wwDBLookupCombo4: TwwDBLookupCombo;
wwIncrementalSearch4: TwwIncrementalSearch;
wwKeyCombo4: TwwKeyCombo;
Label7: TLabel;
Label8: TLabel;
wwDBGrid5: TwwDBGrid;
wwDBLookupCombo5: TwwDBLookupCombo;
wwIncrementalSearch5: TwwIncrementalSearch;
wwKeyCombo5: TwwKeyCombo;
Label9: TLabel;
Label10: TLabel;
wwTable2: TwwTable;
wwDataSource2: TwwDataSource;
wwTable3: TwwTable;
wwDataSource3: TwwDataSource;
wwTable4: TwwTable;
wwDataSource4: TwwDataSource;
wwTable5: TwwTable;
wwDataSource5: TwwDataSource;
wwTable4Language: TStringField;
wwTable4LngProficiency: TStringField;
wwTable4NAME_PERS: TStringField;
wwTable4BEST_LANGUAGE_CIV: TStringField;
wwTable4TOT_CR_HRS_ACAD: TSmallintField;
wwTable4TYPE_CR_HRS_AGRGT: TSmallintField;
wwTable4EDUCATION_AREA_HIGH: TStringField;
wwTable4EDUC_LEVEL_CIV_ENTRY: TStringField;
wwDBLookupCombo6: TwwDBLookupCombo;
wwTable7: TwwTable;
wwTable8: TwwTable;
wwDataSource7: TwwDataSource;
wwDataSource8: TwwDataSource;
wwTable3NAME_PERS: TStringField;
wwTable3SSAN_EMPL_CON_NR: TStringField;
wwTable3DT_LATEST_ENT_PRES_GR: TStringField;
wwTable3DT_LAST_PROM: TStringField;
wwTable3DT_TEMP_PROM_EXPIR: TStringField;
wwTable3GR_CIV_PERM: TStringField;
wwTable3PAY_PLAN_PERM: TStringField;
wwTable3COMP_LEV_PERM: TStringField;
wwTable3OCUPTNL_SRS_PERM: TStringField;
wwTable3STEP_IN_GRADE_CIV: TStringField;
wwTable3PAY_RATE_DETERM: TStringField;
wwTable3DT_WGI_DUE: TStringField;
wwTable3FEGLI: TStringField;
wwTable3HEALTH_PLANS: TStringField;
wwTable3HEALTH_ENRLM: TStringField;
wwTable3DT_FEHB_EFF: TStringField;
wwTable3DT_TEMP_ELIG_FEHB: TStringField;
```

```
    wwTable3DT_LWOP_EXPIR: TStringField;
    wwTable3DT_LWOP_SU_FUR_BEG: TStringField;
    wwTable3BASIC_SALARY_RATE: TIntegerField;
    wwTable3LOC_ADJ: TIntegerField;
    wwTable3ADJ_BASIC_PAY: TIntegerField;
    wwTable3TOTAL_SALARY: TIntegerField;
    wwTable3FROZEN_SERVICE: TSmallintField;
    wwTable3PREV_RETIREMENT_COVG: TStringField;
    wwTable3Rate2: TCurrencyField;
    wwTable3Total2: TCurrencyField;
    wwTable3Loc2: TCurrencyField;
    KingPopup1: TKingPopup;
    wwDBComboDlg1: TwwDBComboDlg;
    wwTable1NAME_PERS: TStringField;
    wwTable1SSAN_EMPL_CON_NR: TStringField;
    wwTable1LOCAL_ADDR_STREET: TStringField;
    wwTable1LOCAL_ADDR_CITY: TStringField;
    wwTable1LOCAL_ADDR_STATE: TStringField;
    wwTable1ADRS_MAIL_ZIP: TStringField;
    wwTable1HANDCP_RPRTBL: TSmallintField;
    wwTable1SEX: TStringField;
    wwTable1RACE_NATIONAL_ORIGIN: TStringField;
    wwTable1CITIZENSHIP: TSmallintField;
    wwTable1FAM_MBR_EMPL_PREF: TStringField;
    wwTable1DOB: TStringField;
    wwTable1CNTY_WORLD_CIT: TStringField;
    wwTable1Race2: TStringField;
    wwTable1Birth2: TStringField;
    wwDBLookupCombo7: TwwDBLookupCombo;
    wwTable9: TwwTable;
    wwDataSource9: TwwDataSource;
    wwDBComboDlg2: TwwDBComboDlg;
    wwDBComboDlg3: TwwDBComboDlg;
    wwDBComboDlg4: TwwDBComboDlg;
    wwDBComboDlg5: TwwDBComboDlg;
    wwDBComboDlg6: TwwDBComboDlg;
    wwDBLookupCombo8: TwwDBLookupCombo;
    wwDBLookupCombo9: TwwDBLookupCombo;
    wwTable10: TwwTable;
    wwTable11: TwwTable;
    wwTable12: TwwTable;
    procedure TabSet1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure wwTable4CalcFields(DataSet: TDataset);
    procedure wwTable3CalcFields(DataSet: TDataset);
    procedure wwDBComboDlg1CustomDlg(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure TabSet1Change(Sender: TObject; NewTab: Integer;
      var AllowChange: Boolean);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  Form2: TForm2;

implementation

{$R *.DFM}

procedure TForm2.TabSet1Change(Sender: TObject; NewTab: Integer;
  var AllowChange: Boolean);
begin
 case TabSet1.TabIndex of
 0: begin
      wwTable2.GotoCurrent(wwTable1);
      wwTable3.GotoCurrent(wwTable1);
      wwTable4.GotoCurrent(wwTable1);
      wwTable5.GotoCurrent(wwTable1);
    end;
 1: begin
      wwTable1.GotoCurrent(wwTable2);
      wwTable3.GotoCurrent(wwTable2);
      wwTable4.GotoCurrent(wwTable2);
      wwTable5.GotoCurrent(wwTable2);
    end;
 2: begin
      wwTable1.GotoCurrent(wwTable3);
      wwTable2.GotoCurrent(wwTable3);
      wwTable4.GotoCurrent(wwTable3);
      wwTable5.GotoCurrent(wwTable3);
    end;
 3: begin
      wwTable1.GotoCurrent(wwTable4);
      wwTable2.GotoCurrent(wwTable4);
      wwTable3.GotoCurrent(wwTable4);
      wwTable5.GotoCurrent(wwTable4);
    end;
 4: begin
      wwTable1.GotoCurrent(wwTable5);
      wwTable2.GotoCurrent(wwTable5);
      wwTable3.GotoCurrent(wwTable5);
      wwTable4.GotoCurrent(wwTable5);
    end;
 end;{Case}
end;

procedure TForm2.TabSet1Click(Sender: TObject);
begin
 Notebook1.PageIndex := TabSet1.TabIndex;
end;



procedure TForm2.FormCreate(Sender: TObject);
begin
  TabSet1.Tabs := Notebook1.Pages;
```

```
  Notebook1.PageIndex := TabSet1.TabIndex;
end;

procedure TForm2.wwTable4CalcFields(DataSet: TDataset);
var
  S : String[6];
begin
  S := wwTable4BEST_LANGUAGE_CIV.value;
  wwTable4Language.Value := Copy(s,1,2);
  wwTable4LngProficiency.Value  := Copy(s,3,1);
end;

procedure TForm2.wwTable3CalcFields(DataSet: TDataset);
begin
  wwTable3Rate2.Value := (wwTable3BASIC_SALARY_RATE.Value/100);
  wwTable3Total2.Value := (wwTable3TOTAL_SALARY.Value/100);
  wwTable3Loc2.Value := (wwTable3LOC_ADJ.Value);
end;

procedure TForm2.wwDBComboDlg1CustomDlg(Sender: TObject);
begin
  KingPopup1.AlignSource := (Sender as TwwDBComboDlg);
  KingPopup1.Execute;
end;


procedure TForm2.BitBtn1Click(Sender: TObject);
begin
  Form2.Close;
end;

end.
```



File Name: UEDITOR.PAS

```
unit Ueditor;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Wwkeycb, DBCtrls, DB, DBTables, Wwtable,
  Wwdatsrc, ExtCtrls, Buttons, Tabs, wwdblook, Mask, Wwdbcomb, KingPop,
  EPCalPop;

type
  TfrmEditor = class(TForm)
    Panel1: TPanel;
    TabSet1: TTabSet;
    Bevel1: TBevel;
    Bevel2: TBevel;
    BitBtn1: TBitBtn;
    Notebook1: TNotebook;
    DBText1: TDBText;
    DBText2: TDBText;
    wwDataSource1: TwwDataSource;
```

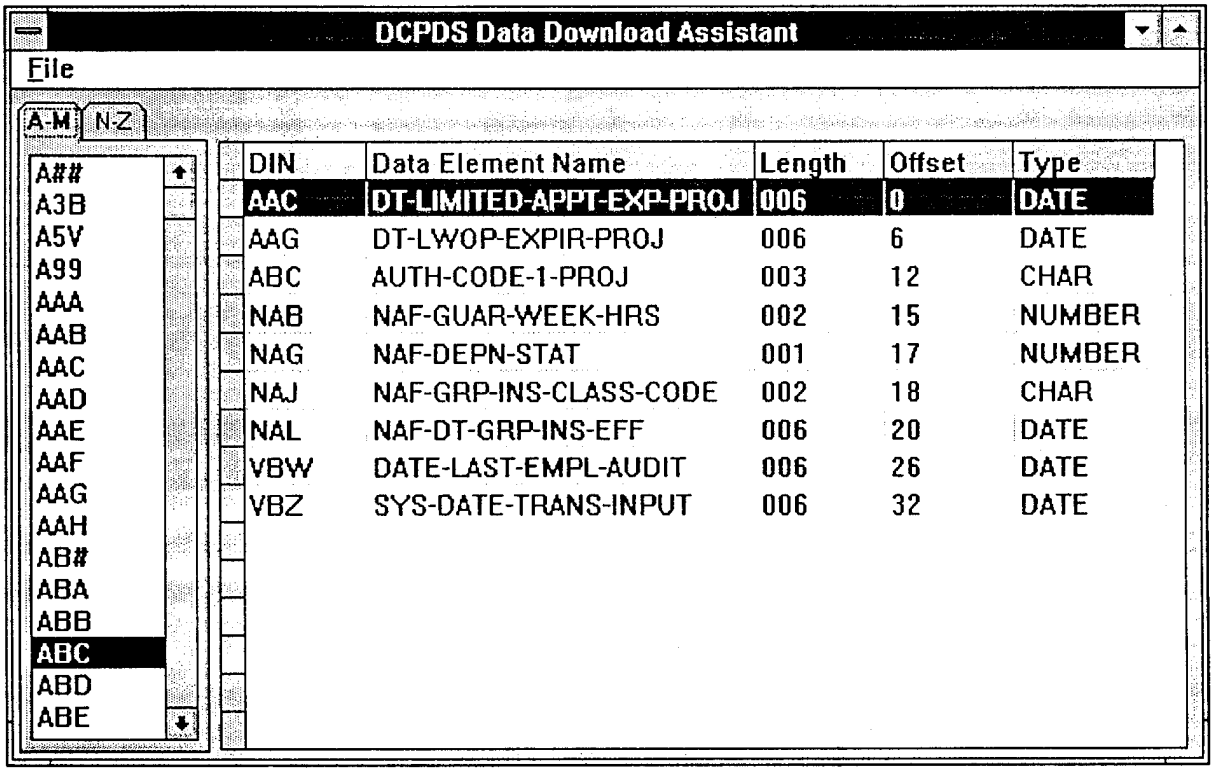
```
wwTable1: TwwTable;
GroupBox1: TGroupBox;
wwIncrementalSearch1: TwwIncrementalSearch;
Image1: TImage;
wwTable1NAME_PERS: TStringField;
wwTable1SSAN_EMPL_CON_NR: TStringField;
wwTable1DT_FEHB_REG_ELIG_EXPIR_P: TStringField;
wwTable1DRAWDOWN_ACTION_ID_PROJ: TStringField;
wwTable1PERS_SCTY_CLEAR_ELIG: TStringField;
wwTable1PART_TIME_INDCTR_PROJ: TStringField;
wwTable1LOCAL_ADDR_STREET: TStringField;
wwTable1LOCAL_ADDR_CITY: TStringField;
wwTable1LOCAL_ADDR_STATE: TStringField;
wwTable1ADRS_MAIL_ZIP: TStringField;
wwTable1BEST_LANGUAGE_CIV: TStringField;
wwTable1TOT_CR_HRS_ACAD: TSmallintField;
wwTable1TYPE_CR_HRS_AGRGT: TSmallintField;
wwTable1EDUCATION_AREA_HIGH: TStringField;
wwTable1EDUC_LEVEL_CIV_ENTRY: TStringField;
wwTable1DT_LATEST_ENT_PRES_GR: TStringField;
wwTable1DT_LAST_PROM: TStringField;
wwTable1DT_TEMP_PROM_EXPIR: TStringField;
wwTable1GR_CIV_PERM: TStringField;
wwTable1PAY_PLAN_PERM: TStringField;
wwTable1COMP_LEV_PERM: TStringField;
wwTable1OCUPTNL_SRS_PERM: TStringField;
wwTable1HANDCP_RPRTBL: TSmallintField;
wwTable1HRS_SCH_WEEK: TSmallintField;
wwTable1STA_PREC_EMPLM: TSmallintField;
wwTable1DT_PROB_TRI_PRD_BEG: TStringField;
wwTable1DT_SUP_MGR_PROB_ENDS: TStringField;
wwTable1SUPV_MGR_PROBN_COMPLTION: TStringField;
wwTable1DRAWDOWN_ACTION_ID: TStringField;
wwTable1AGCY_CD_TRANS_FR: TStringField;
wwTable1DT_TVL_AGRMT_PCS_EXPIR: TStringField;
wwTable1POSN_GR_CIV: TStringField;
wwTable1PROG_ELEMENT: TStringField;
wwTable1ORG_STRUCT_ID_SHRED: TStringField;
wwTable1DRUG_TEST_RQD: TStringField;
wwTable1CPCN: TStringField;
wwTable1SUPV_STATUS: TSmallintField;
wwTable1BARG_UNIT_STAT: TSmallintField;
wwTable1POSN_TITLE: TStringField;
wwTable1CURR_PAY_PLAN: TStringField;
wwTable1OCUPTNL_SRS: TStringField;
wwTable1COMP_LEV: TStringField;
wwTable1FLSA_CAT: TStringField;
wwTable1POSN_WRK_SCHED: TStringField;
wwTable1STEP_IN_GRADE_CIV: TStringField;
wwTable1PAY_RATE_DETERM: TStringField;
wwTable1DT_WGI_DUE: TStringField;
wwTable1FEGLI: TStringField;
wwTable1HEALTH_PLANS: TStringField;
wwTable1HEALTH_ENRLM: TStringField;
wwTable1DT_FEHB_EFF: TStringField;
```

```
wwTable1DT_TEMP_ELIG_FEHB: TStringField;
wwTable1DT_LWOP_EXPIR: TStringField;
wwTable1DT_LWOP_SU_FUR_BEG: TStringField;
wwTable1BASIC_SALARY_RATE: TIntegerField;
wwTable1LOC_ADJ: TIntegerField;
wwTable1ADJ_BASIC_PAY: TIntegerField;
wwTable1TOTAL_SALARY: TIntegerField;
wwTable1FROZEN_SERVICE: TSmallintField;
wwTable1PREV_RETIREMENT_COVG: TStringField;
wwTable1SEX: TStringField;
wwTable1RACE_NATIONAL_ORIGIN: TStringField;
wwTable1CITIZENSHIP: TSmallintField;
wwTable1FAM_MBR_EMPL_PREF: TStringField;
wwTable1DOB: TStringField;
wwTable1CNTY_WORLD_CIT: TStringField;
wwTable1NTR_ACTION_PERS: TStringField;
wwTable1LAST_AUTH_CD_1: TStringField;
wwTable1LAST_AUTH_CD_2: TStringField;
wwTable1RETIREMENT_PLAN: TStringField;
wwTable1FERS_COVERAGE: TStringField;
wwTable1ANNUITANT_INDICATOR: TStringField;
wwTable1DATE_EOD_CURR_AGCY: TStringField;
wwTable1DT_ARR_SVCG_CCPO: TStringField;
wwTable1DT_CONV_CAR_BEG: TStringField;
wwTable1DT_CONV_REC_DUE: TStringField;
wwTable1DT_REC_CONV_BEG: TStringField;
wwTable1DT_RTND_GRADE_BEG: TStringField;
wwTable1DT_RTND_GRADE_EXPIR: TStringField;
wwTable1RTND_PAY_PLAN: TStringField;
wwTable1RTND_GR_CIV: TStringField;
wwTable1RTND_OCUPTNL_SRS: TStringField;
wwTable1RTND_STEP_IN_GRADE_CIV: TStringField;
wwTable1RTND_PAY_TABLE_IDENT: TStringField;
wwTable1RTND_PAY_BASIS: TStringField;
wwTable1RTND_GRADE_FLAG_PROJ: TStringField;
wwTable1RTND_LOCALITY_PCT: TSmallintField;
wwTable1VET_PREF_APPT: TSmallintField;
wwTable1VET_PREF_RIF: TSmallintField;
wwTable1TENURE_GP_EMPL: TSmallintField;
wwTable1APPT_TYPE: TStringField;
wwTable1RESERVE_CATEGORY: TStringField;
wwTable1SCD_CIV_LEAVE: TStringField;
wwTable1SCD_CIV_RIF: TStringField;
wwTable1DT_LAST_EQUIV_INC: TStringField;
wwTable1DT_TEMP_APPT_EXPIR: TStringField;
wwTable1DT_VET_REAJMNT_CONV_DUE: TStringField;
wwTable1SCD_CIV: TStringField;
wwTable1DT_TEMP_REASMT_EXP: TStringField;
wwTable1CREDITABLE_MIL_SVC: TSmallintField;
wwTable1ORIG_APPT_AUTH_CD_1: TStringField;
wwTable1DT_RET_UNFM_SVC: TStringField;
wwTable1UNIF_SVS_COMP: TSmallintField;
wwTable1UNIF_SVC_DESIGN: TStringField;
wwTable1GR_RET: TSmallintField;
wwTable1MIL_RET_WAV_IND: TStringField;
```

```
wwTable1EXCP_RETM_PAY_IND: TSmallintField;
wwTable1VETERANS_STATUS: TStringField;
wwTable1CCPO_SUSP_CD_1: TStringField;
wwTable1CCPO_SUSP_DT_1: TStringField;
wwTable1CCPO_SUSP_CD_2: TStringField;
wwTable1CCPO_SUSP_DT_2: TStringField;
wwTable1TYPE_OF_EMPLMENT: TStringField;
wwTable1TSP_STATUS: TStringField;
wwTable1TSP_STATUS_DATE: TStringField;
wwTable1TSP_SCD: TStringField;
wwTable1TSP_RATE: TSmallintField;
wwTable1TSP_EMPL_AMT: TSmallintField;
wwTable1TSP_ELIGIBILITY_DATE: TStringField;
wwTable1OBL_CPCN_FLAG: TStringField;
wwTable1NV_LOGIC_ORGN_CODE: TStringField;
wwTable1NV_CIT_BASIS: TStringField;
wwTable1NV_LOCAL_EMP_ID_NR: TStringField;
wwTable1NV_APPROP_CD: TStringField;
wwTable1CCPO_FOREIGN_COUNTRY: TStringField;
wwTable1CCPO_VISA_TYPES: TStringField;
wwTable1CCPO_VISA_EXP_DATES: TStringField;
wwTable1CCPO_SPOUSE_EMP_PRO: TStringField;
Panel2: TPanel;
wwDBComboBox1: TwwDBComboBox;
DBEdit1: TDBEdit;
Label1: TLabel;
Label2: TLabel;
wwDBLookupCombo1: TwwDBLookupCombo;
wwTable2: TwwTable;
Label3: TLabel;
Label5: TLabel;
DBCheckBox1: TDBCheckBox;
wwDBLookupCombo2: TwwDBLookupCombo;
wwTable3: TwwTable;
GroupBox2: TGroupBox;
DBEdit5: TDBEdit;
DBEdit2: TDBEdit;
DBEdit4: TDBEdit;
DBEdit3: TDBEdit;
Edit1: TEdit;
Label6: TLabel;
GroupBox3: TGroupBox;
wwDBLookupCombo4: TwwDBLookupCombo;
Label7: TLabel;
Label8: TLabel;
ComboBox1: TComboBox;
GroupBox4: TGroupBox;
Label4: TLabel;
Label10: TLabel;
Label12: TLabel;
Label13: TLabel;
Label14: TLabel;
Label15: TLabel;
Label16: TLabel;
Label17: TLabel;
```

```
    Label18: TLabel;
    DBEdit6: TDBEdit;
    DBEdit7: TDBEdit;
    DBEdit8: TDBEdit;
    DBEdit9: TDBEdit;
    DBEdit10: TDBEdit;
    DBEdit11: TDBEdit;
    DBEdit12: TDBEdit;
    DBEdit13: TDBEdit;
    DBEdit14: TDBEdit;
    Label9: TLabel;
    Label11: TLabel;
    DBEdit15: TDBEdit;
    DBEdit16: TDBEdit;
    GroupBox5: TGroupBox;
    Label19: TLabel;
    Label20: TLabel;
    Label21: TLabel;
    Label22: TLabel;
    Label23: TLabel;
    Label24: TLabel;
    Label25: TLabel;
    Label26: TLabel;
    Label27: TLabel;
    Label28: TLabel;
    EPCalendarPopup1: TEPCalendarPopup;
    SpeedButton1: TSpeedButton;
    procedure FormCreate(Sender: TObject);
    procedure TabSet1Click(Sender: TObject);
    procedure wwDataSource1DataChange(Sender: TObject; Field: TField);
    procedure SpeedButton1Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmEditor: TfrmEditor;

implementation

{$R *.DFM}

procedure TfrmEditor.FormCreate(Sender: TObject);
begin
  TabSet1.Tabs := Notebook1.Pages;
  Notebook1.PageIndex := TabSet1.TabIndex;
end;

procedure TfrmEditor.TabSet1Click(Sender: TObject);
begin
  Notebook1.PageIndex := TabSet1.TabIndex;
end;
```

```
procedure TfrmEditor.wwDataSource1DataChange(Sender: TObject; Field: TField);
begin
 if wwTable1NAME_PERS.AsString = 'AGUILAR MARY M' then
   begin
     Image1.Visible := true;
     Edit1.Text := 'Keep Guessing';
   end
 else
   begin
     Image1.Visible := False;
     Edit1.Text := '';
   end;
end;

procedure TfrmEditor.SpeedButton1Click(Sender: TObject);
var
  P: TPoint;
begin
  with (Sender as TSpeedButton) do
    P := Point(Left, Top + Height - 1);
  EPCalendarPopup1.Popup(Date, P.X, P.Y);
end;

procedure TfrmEditor.BitBtn1Click(Sender: TObject);
begin
 close;
end;

end.
```



File Name: UDESIRES.PAS

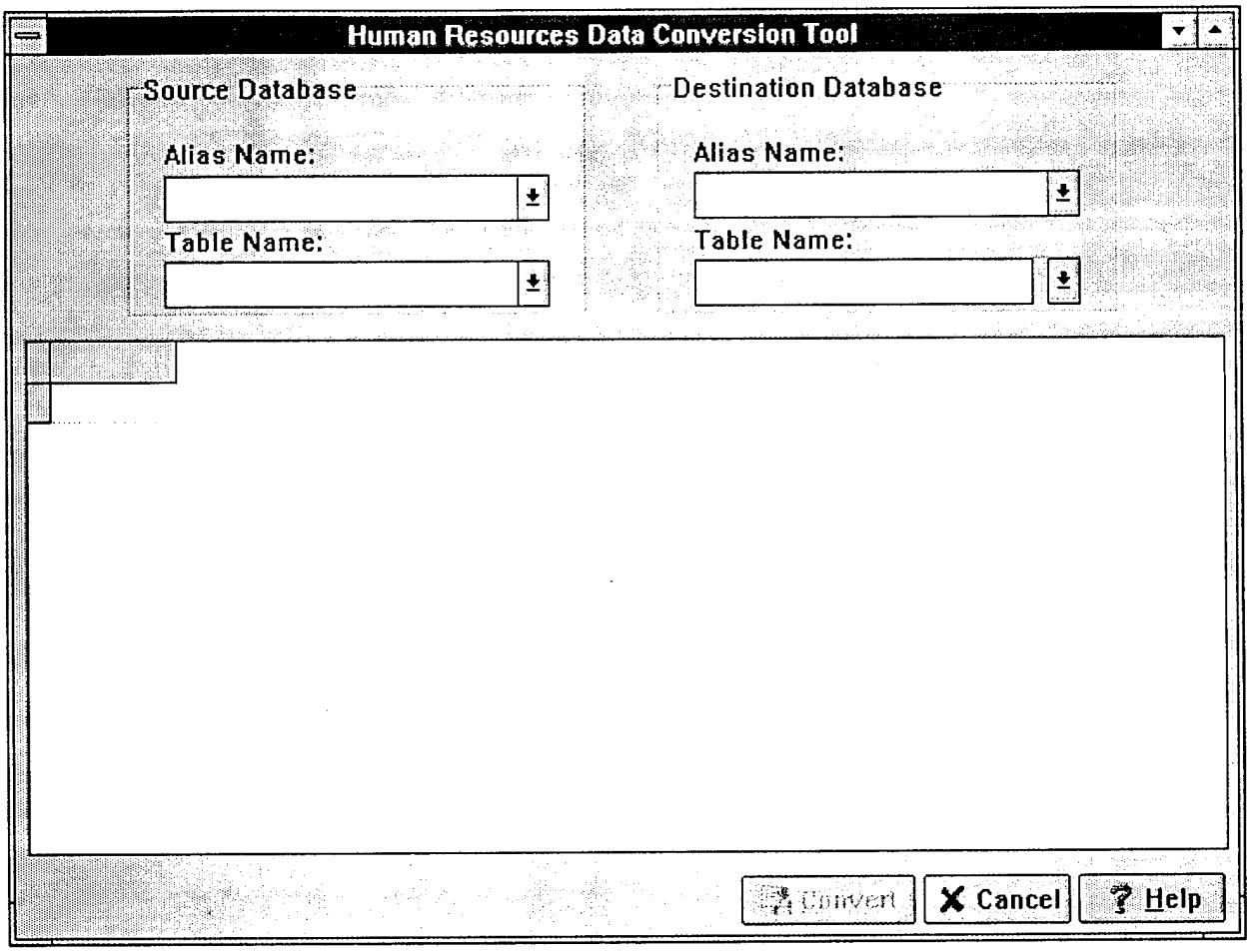
```
unit Udesires;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, DB, DBTables, Grids, DBGrids, DBLookup, StdCtrls,
  TabNotBk, Menus;

type
  TfrmDesire = class(TForm)
    Table1: TTable;
    Table1DIN: TStringField;
    TabbedNotebook1: TTabbedNotebook;
    ListBox1: TListBox;
    ListBox2: TListBox;
    StringGrid1: TStringGrid;
    Table1DATA_NAME: TStringField;
    Table1FSV: TStringField;
    MainMenu1: TMainMenu;
    N1: TMenuItem;
    N2: TMenuItem;
    N3: TMenuItem;
    N4: TMenuItem;
    N5: TMenuItem;
```

```
    N6: TMenuItem;
    N7: TMenuItem;
    N8: TMenuItem;
    O1: TMenuItem;
    Table1C: TStringField;
    procedure ListBox1GetItem(Sender: TObject; Index: Longint;
      var ItemString: OpenString);
    procedure FormCreate(Sender: TObject);
    procedure ListBox1DblClick(Sender: TObject);
    procedure N2Click(Sender: TObject);
    procedure D1Click(Sender: TObject);
    procedure P1Click(Sender: TObject);
    procedure N7Click(Sender: TObject);
    procedure O1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmDesire: TfrmDesire;

implementation

{$R *.DFM}

uses
 UCPDSFrm,
 UMerge;

procedure TfrmDesire.ListBox1GetItem(Sender: TObject; Index: Longint;
  var ItemString: OpenString);
begin
  Table1.First;
  while not Table1.EOF do
  begin
    ItemString := Table1.FieldByName('DIN').AsString;
    Table1.Next;
  end;
end;

procedure TfrmDesire.FormCreate(Sender: TObject);
begin
  Table1.First;
  while (Table1.FieldByName('DIN').AsString < 'NAA') do
    begin
      ListBox1.Items.Add(Table1.FieldByName('DIN').AsString);
      Table1.Next;
    end;
  while (Table1.FieldByName('DIN').AsString < 'Y##') do
    begin
      ListBox2.Items.Add(Table1.FieldByName('DIN').AsString);
      Table1.Next;
    end;
```

```
  StringGrid1.Cells[1,0] := 'DIN';
  StringGrid1.Cells[2,0] := 'Data Element Name';
  StringGrid1.Cells[3,0] := 'Length';
  StringGrid1.Cells[4,0] := 'Offset';
  StringGrid1.Cells[5,0] := 'Type';
end;

procedure TfrmDesire.ListBox1DblClick(Sender: TObject);
var
  FieldType: String[10];
  C: String[2];
begin
   Table1.First;
   if StringGrid1.Tag = StringGrid1.RowCount then
     begin
     StringGrid1.RowCount := StringGrid1.RowCount + 1;
     StringGrid1.TopRow := StringGrid1.TopRow + 1;
     StringGrid1.Row := StringGrid1.RowCount - 1;
     end;
   StringGrid1.Cells[1, StringGrid1.Tag] :=
        (Sender as TListBox).Items[(Sender as TListBox).ItemIndex];
   Table1.FindKey([StringGrid1.Cells[1, StringGrid1.Tag]]);
   StringGrid1.Cells[2, StringGrid1.Tag] :=
Table1.FieldByName('DATA_NAME').AsString;
   StringGrid1.Cells[3, StringGrid1.Tag] :=
Table1.FieldByName('FSV').AsString;

   if StringGrid1.Tag = 1 then StringGrid1.Cells[4, StringGrid1.Tag] := '0'
   else
      StringGrid1.Cells[4, StringGrid1.Tag] :=
          IntToStr(StrToInt(StringGrid1.Cells[4, (StringGrid1.Tag - 1)]) +
                   StrToInt(StringGrid1.Cells[3, (StringGrid1.Tag - 1)]));
   C := Table1.FieldByName('C').AsString;
   if  C = 'A' then FieldType := 'CHAR';
   if  C = 'X' then FieldType := 'CHAR';
   if  C = 'D' then FieldType := 'DATE';
   if  C = 'N' then
     if (Table1.FieldByName('FSV').AsInteger) < 5 then
        FieldType := 'NUMBER'
     else
        FieldType := 'LONGINT';
   StringGrid1.Cells[5, StringGrid1.Tag] := FieldType;
   StringGrid1.Tag := StringGrid1.Tag + 1;
end;

procedure TfrmDesire.N2Click(Sender: TObject);
var
i: integer;
begin
   if MessageDlg('This Will Remove All Data' + Chr(13) +
                 'From The Current Project.' + Chr(13) + '          Continue?',
                  mtWarning, [mbYes, mbNo], 0) = mrYes then
      begin
        StringGrid1.RowCount := 16;
        StringGrid1.Tag := 1;
```

```
        for i := 1 to StringGrid1.RowCount do
        begin
          StringGrid1.Cells[1, i] := '';
          StringGrid1.Cells[2, i] := '';
          StringGrid1.Cells[3, i] := '';
          StringGrid1.Cells[4, i] := '';
          StringGrid1.Cells[5, i] := '';
        end;
      end;
end;

procedure TfrmDesire.D1Click(Sender: TObject);
var
  Header: String[125];
  S: String[10];
  P: PChar;
  F1: TextFile;
  I: Integer;
  DINList: TStringList;
begin
frmDesireName := TfrmDesireName.Create(Self);
 try
  frmDesireName.Edit2.Text := IntToStr(StrToInt(StringGrid1.Cells[3,
(StringGrid1.Tag - 1)]) +
                              StrToInt(StringGrid1.Cells[4, (StringGrid1.Tag
- 1)]) + 10);
  frmDesireName.ShowModal;
  if frmDesireName.ModalResult = mrOK then
  begin
  Header := 'DESIRE.ID POSN-MGT RK FOR FRANKIE.PERS RECD CA CCPO QH.SL IF ' +
            'BAA > " ".OT DISK LABEL ' + frmDesireName.Edit1.Text + ' RECD ' +
            frmDesireName.Edit2.Text + ' BLOC 1.WR ';
  DINList := TStringList.Create;
  DINList.Add(Header);
  for I := 1 to (StringGrid1.Tag -1) do
    begin
      S := IntToStr(StrToInt(StringGrid1.Cells[4, i]) + 1) + ' ' +
                    StringGrid1.Cells[1, i] + ' ';
      DINList.Add(S);
    end;
  DINList.SaveToFile(frmDesireName.Edit1.Text + '.dsr');
  end;
 finally
  frmDesireName.Free;
  DINList.Free;
 end;
end;

procedure TfrmDesire.P1Click(Sender: TObject);
var
  Header: String[30];
  S: String[4];
  L: String[4];
  I : Integer;
  DINList: TStringList;
```

```
begin
  DINList := TStringList.Create;
  frmSchemaName := TfrmSchemaName.Create(Self);
  try
    frmSchemaName.ShowModal;
    if frmSchemaName.ModalResult = mrOK then
    begin
    DINList.Add('['+frmSchemaName.Edit1.Text+']');
    DINList.Add('FILETYPE = fixed');
    DINList.Add('CHARSET = ascii');
    DINList.Add('DELIMITER =');
    DINList.Add('Separator =');
    for I := 1 to (StringGrid1.Tag -1) do
    begin
      S := StringGrid1.Cells[3, I];
      S := Copy(S, 2, 2);
      L := StringGrid1.Cells[4, I];
      if Length(L) < 2 then L := '0' + L;
      DINList.Add('Field'+IntToStr(I)+'='+StringGrid1.Cells[2,I]+','+
                  StringGrid1.Cells[5,I]+','+S+',00,'+ L);
    end;
    DINList.SaveToFile(frmSchemaName.Edit1.Text+'.sch');
    end;
  finally
    DINList.Free;
    frmSchemaName.Free;
  end;
end;

procedure TfrmDesire.O1Click(Sender: TObject);
begin
  D1Click(Sender as TObject);
  P1Click(Sender as Tobject);
end;

procedure TfrmDesire.N7Click(Sender: TObject);
begin
  Close;
end;


end.
```



File Name: UTESTONE.PAS

```
unit Utestone;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, DBTables, DB, Grids, DBGrids, Buttons, FileCtrl;

type
  TfrmMaintenance = class(TForm)
    Source: TTable;
    Destination: TTable;
    BatchMove1: TBatchMove;
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    CancelBtn: TBitBtn;
    HelpBtn: TBitBtn;
    ConvertBtn: TBitBtn;
    SourceGrp: TGroupBox;
    Label1: TLabel;
    SrcAlias: TComboBox;
```

```
    Label2: TLabel;
    SrcTable: TComboBox;
    DestGrp: TGroupBox;
    Label3: TLabel;
    Label4: TLabel;
    DestAlias: TComboBox;
    DestTable: TComboBox;
    FileListBox1: TFileListBox;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure SrcAliasChange(Sender: TObject);
    procedure CancelBtnClick(Sender: TObject);
    procedure SrcTableChange(Sender: TObject);
    procedure DestAliasChange(Sender: TObject);
    procedure DestTableExit(Sender: TObject);
    procedure ConvertBtnClick(Sender: TObject);
    procedure DestTableChange(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmMaintenance: TfrmMaintenance;

implementation

{$R *.DFM}

procedure TfrmMaintenance.Button1Click(Sender: TObject);
begin
  batchmove1.execute;
end;

procedure TfrmMaintenance.FormCreate(Sender: TObject);
begin
  Session.GetAliasNames(SrcAlias.Items);
  DestAlias.Items := SrcAlias.Items;
end;

procedure TfrmMaintenance.SrcAliasChange(Sender: TObject);
begin
  SrcTable.Clear;
  Session.GetAliasParams(SrcAlias.Items[SrcAlias.ItemIndex], SrcTable.Items);
  FileListBox1.Directory := Copy(SrcTable.Items[0] , 6, 30) + '\';
  SrcTable.Items.Clear;
  SrcTable.Items := FileListBox1.Items;
end;

procedure TfrmMaintenance.DestAliasChange(Sender: TObject);
begin
  Session.GetTableNames(DestAlias.Items[DestAlias.ItemIndex], '', True, False,
DestTable.Items);
end;
```

```
procedure TfrmMaintenance.SrcTableChange(Sender: TObject);
var
FileExt : String[3];
begin
  Source.Active := False;
  Source.DatabaseName := SrcAlias.Items[SrcAlias.ItemIndex];
  Source.TableName := SrcTable.Items[SrcTable.ItemIndex];
  FileExt := ExtractFileExt(Source.TableName);
  if FileExt = 'db'  then Source.TableType := ttParadox else
  if FileExt = 'dbf' then Source.TableType := ttDBase else
  if FileExt = 'txt' then Source.TableType := ttASCII else
    Source.TableType := ttDefault;
  Source.Active := True;
end;

procedure TfrmMaintenance.DestTableChange(Sender: TObject);
begin
  if DestTable.Text <> '' then ConvertBtn.Enabled := True;
end;

procedure TfrmMaintenance.DestTableExit(Sender: TObject);
var
FileExt : String[3];
begin
  Destination.DatabaseName := DestAlias.Items[DestAlias.ItemIndex];
  Destination.TableName := DestTable.Text;
  FileExt := ExtractFileExt(Destination.TableName);
  if FileExt = '.db'  then Destination.TableType := ttParadox else
  if FileExt = '.dbf' then Destination.TableType := ttDBase else
  if FileExt = '.txt' then Destination.TableType := ttASCII else
    Destination.TableType := ttParadox;
end;

procedure TfrmMaintenance.ConvertBtnClick(Sender: TObject);
const
  MSG = '   Successful Conversion!'+ Chr(13) +'Return To Main Application?';
begin
  Source.Active := False;
  BatchMove1.Execute;
  Source.Active := True;
  if MessageDlg(MSG, mtInformation, [mbYes, mbNo], 0) = mrYes then
    CancelBtn.Click;
end;

procedure TfrmMaintenance.CancelBtnClick(Sender: TObject);
begin
  Close;
end;

end.
```

LIST OF REFERENCES

1. Waterson, Karen, *Client/Server Technology for Managers*, Addison-Wesley Publishing Company, 1995.

2. Orfali, Robert, Dan Harkley, and Jeri Edwards, *Essential Client/Server Survival Guide*, Van Nostrand Reinhold, 1994.

3. Gartner Group conference presentation, *Client/Server Computing: Where Does It Lead?*, Gartner Group, 1994.

4. Vaskevitch, David, *Client/Server Strategies, A Survival Guide for Corporate Reengineers*, IDG Books Worldwide, Inc., 1993.

5. Kroenke, David M., *Database Processing: Fundamentals, Design, Implementation*, Fourth Edition, MacMillan Publishing Company, 1992.

6. Vaughn, Larry T., *Client Server System Design & Implementation*, McGraw-Hill, Inc., 1994.

7. Taylor, Lloyd, *comp.client-server Frequently Asked Questions*, April 1995.

8. Brooks, Fredrick, *The Mythical Man-Month*, Addison-Wesley Publishing Company, 1989.

9. Date, C. J., *An Introduction To Database Systems*, Third Edition, Addison-Wesley Publishing Company, 1982.

Initial Distribution List

1. Defense Technical Information Center — 2
   Cameron Station
   Alexandria, Virginia 22304-6145

2. Library, Code 52 — 2
   Naval Postgraduate School
   Monterey, CA 93943-5101

3. Magdi N. Kamel, Code SM/Ka — 1
   Department of Systems Management
   Naval Postgraduate School
   Monterey, CA 93943-5002

4. James C. Emery, Code SM/Ey — 1
   Department of Systems Management
   Naval Postgraduate School
   Monterey, CA 93943-5002

5. Mary Aguilar, Director of Human Resources — 1
   Human Resources, Office
   Naval Postgraduate School
   Monterey, CA 93943-5002

6. Charles Calvert, Developer Relations — 1
   Borland International
   100 Borland Way
   Scotts Valley, CA 95066-3249

7. LT David R. McDermitt, USN — 2
   1279 Sixth Street
   Monterey, CA 93940